Summary so far

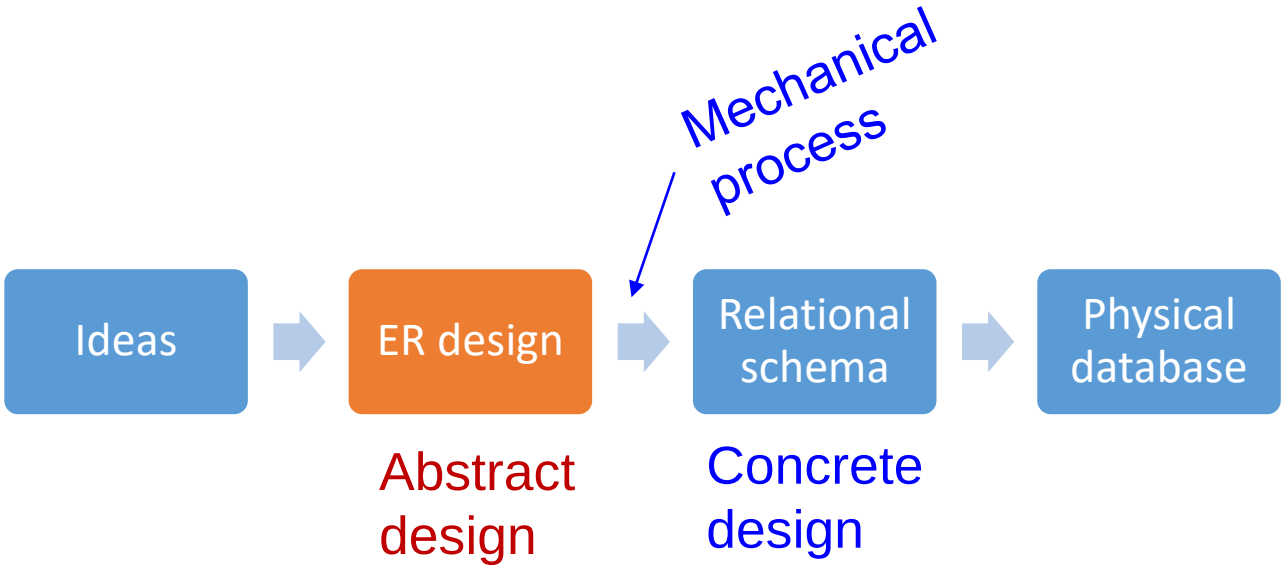
A *data model* is a collection of concepts

A *schema* is a description of data, using data model.

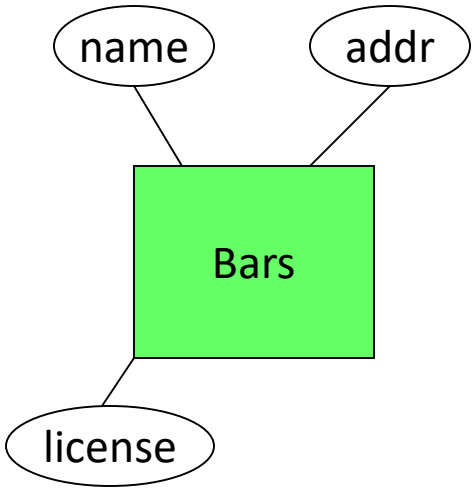
A *database (instance)* is a collection of data compliant with the schema

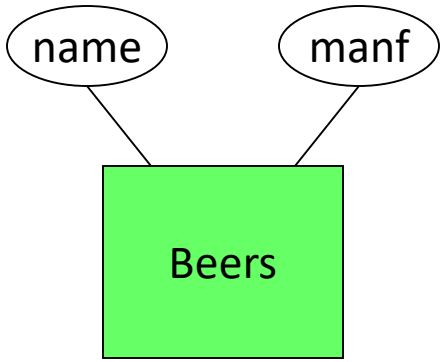
Process of database design

- Notation for expressing designs: Entity-Relationship (E/R) model



Step 1. Identify **entities** (entity sets) and their attributes



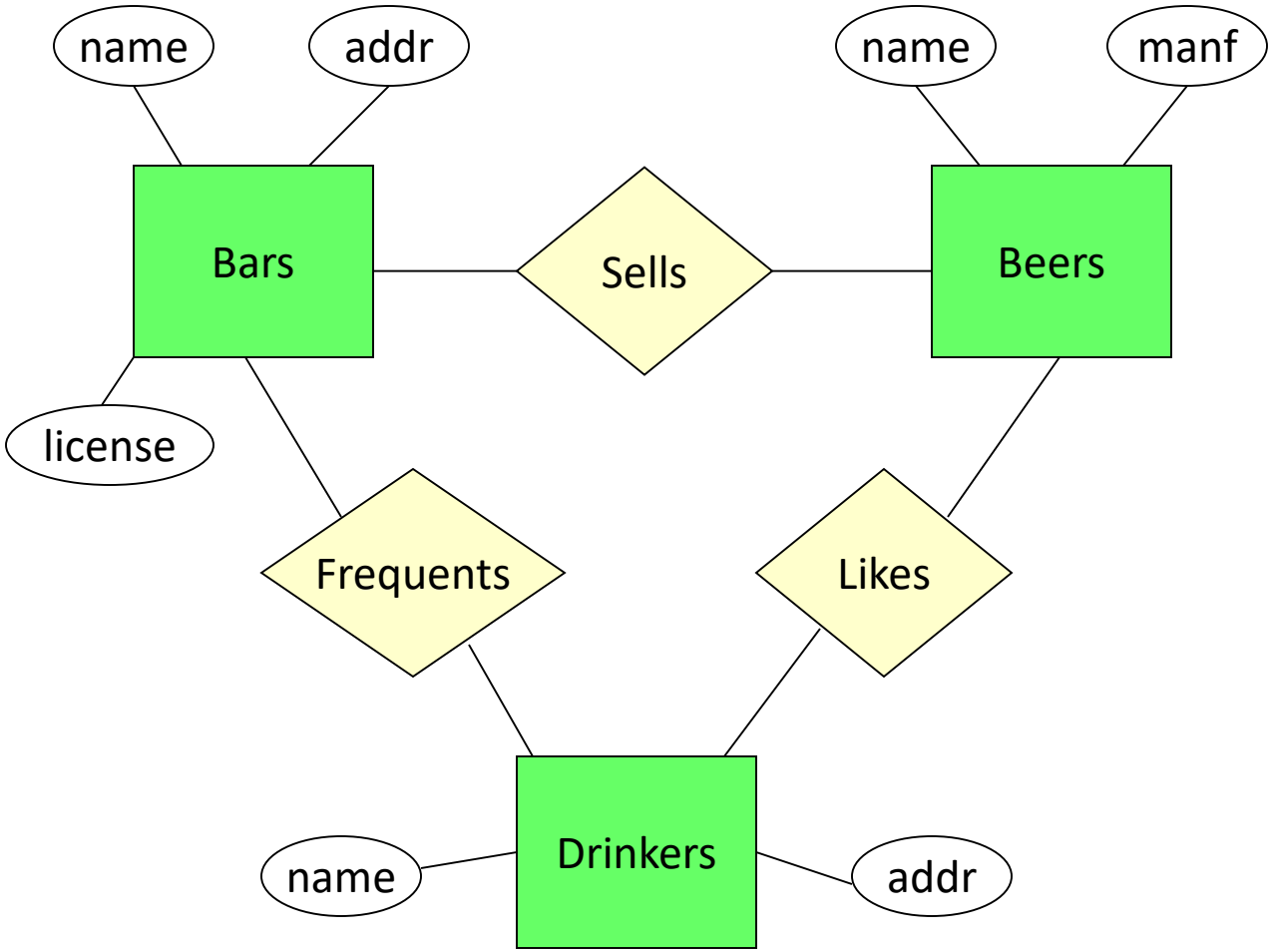


Bars sell some beers.

Drinkers like some beers.

Drinkers frequent some bars.

Step 2. Identify **relationships** between entities (relationship sets) and their attributes

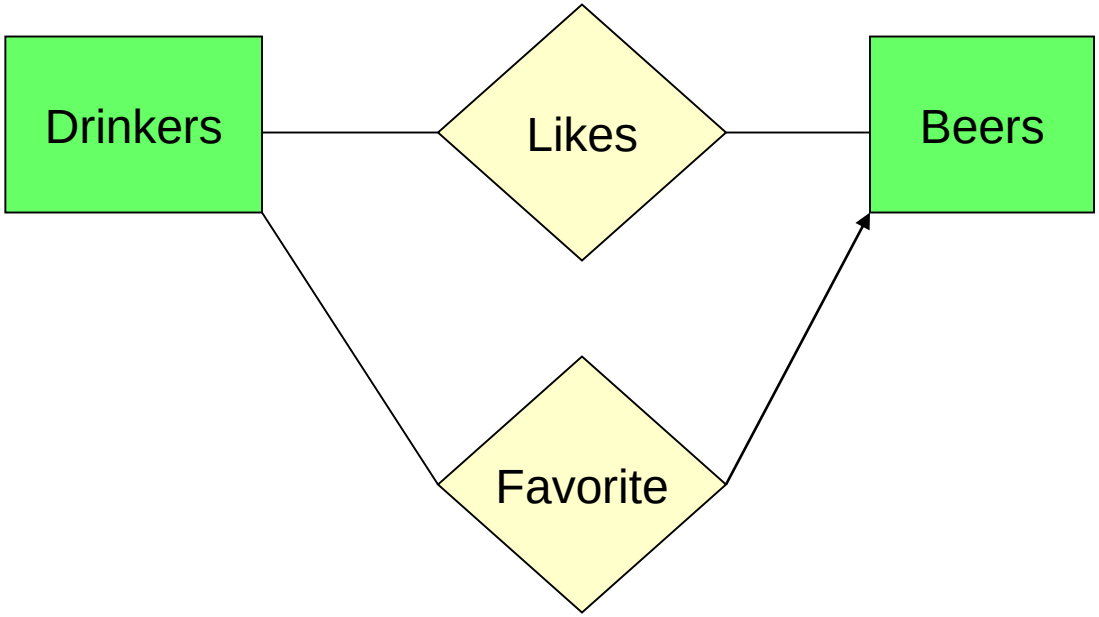


Bars sell some beers.

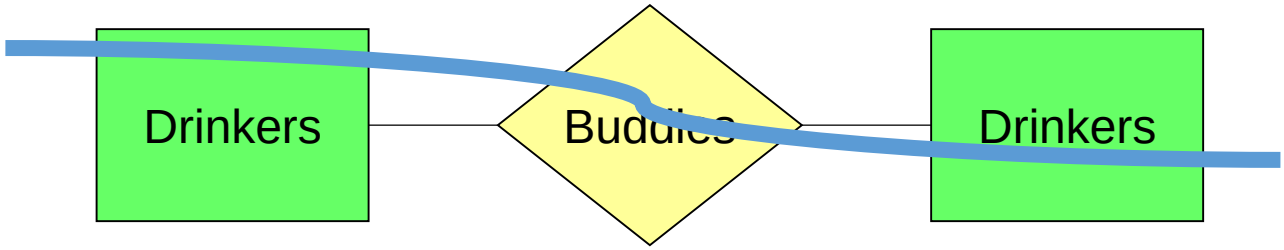
Drinkers like some beers.

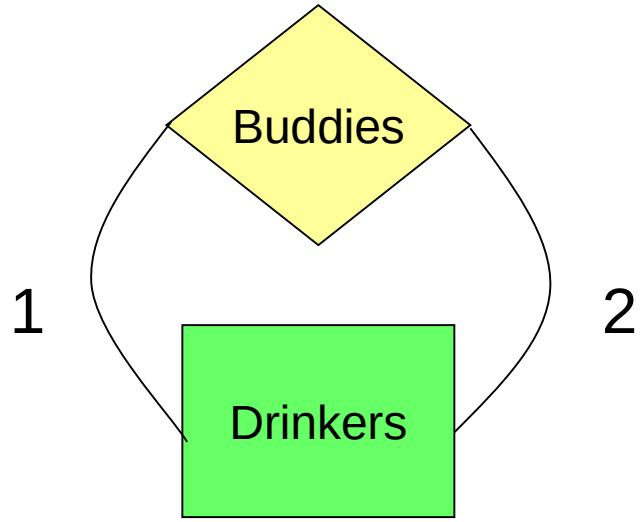
Drinkers frequent some bars.

Multiple relationships may exist between the same two entity sets

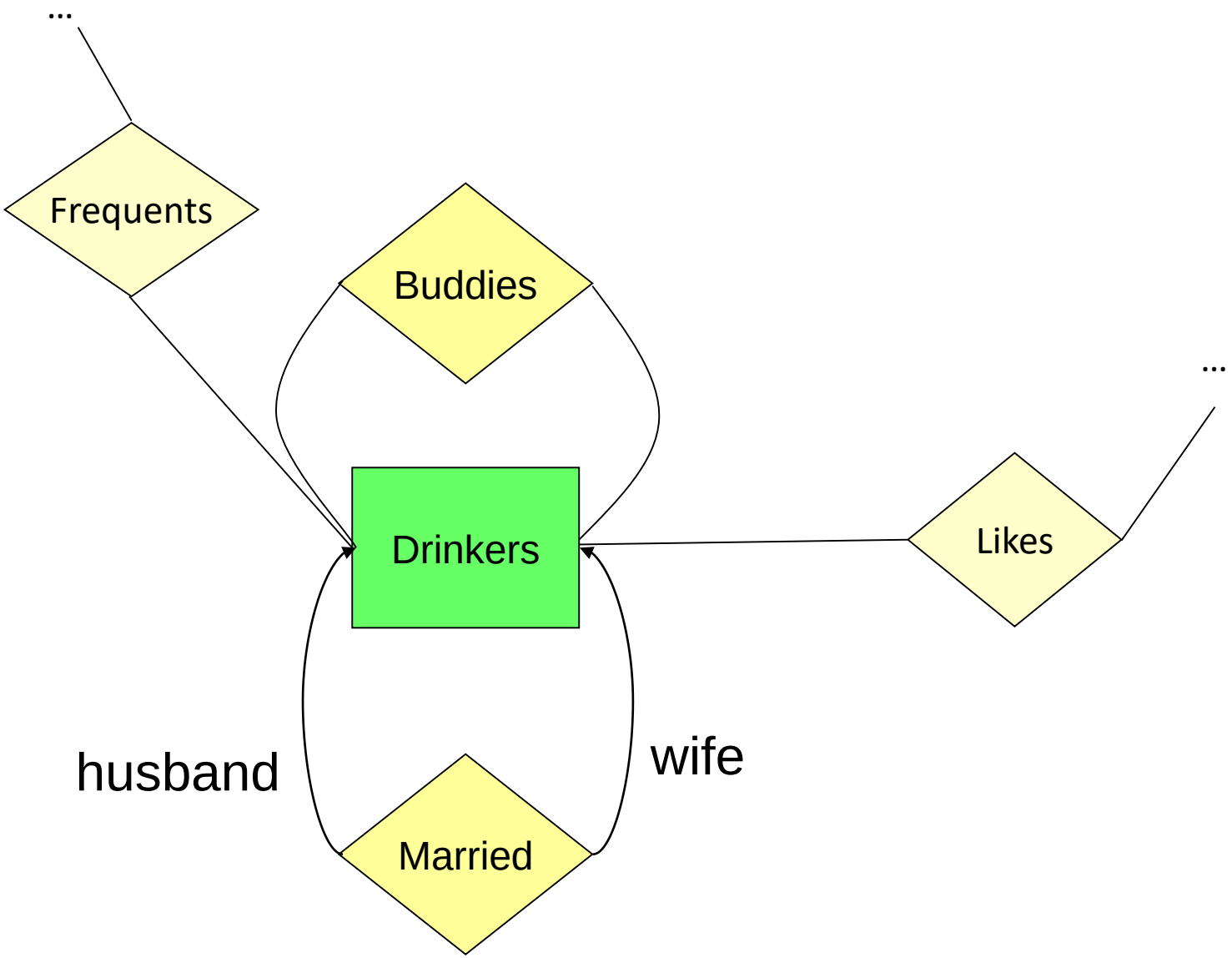


Recursive (self)-relationships

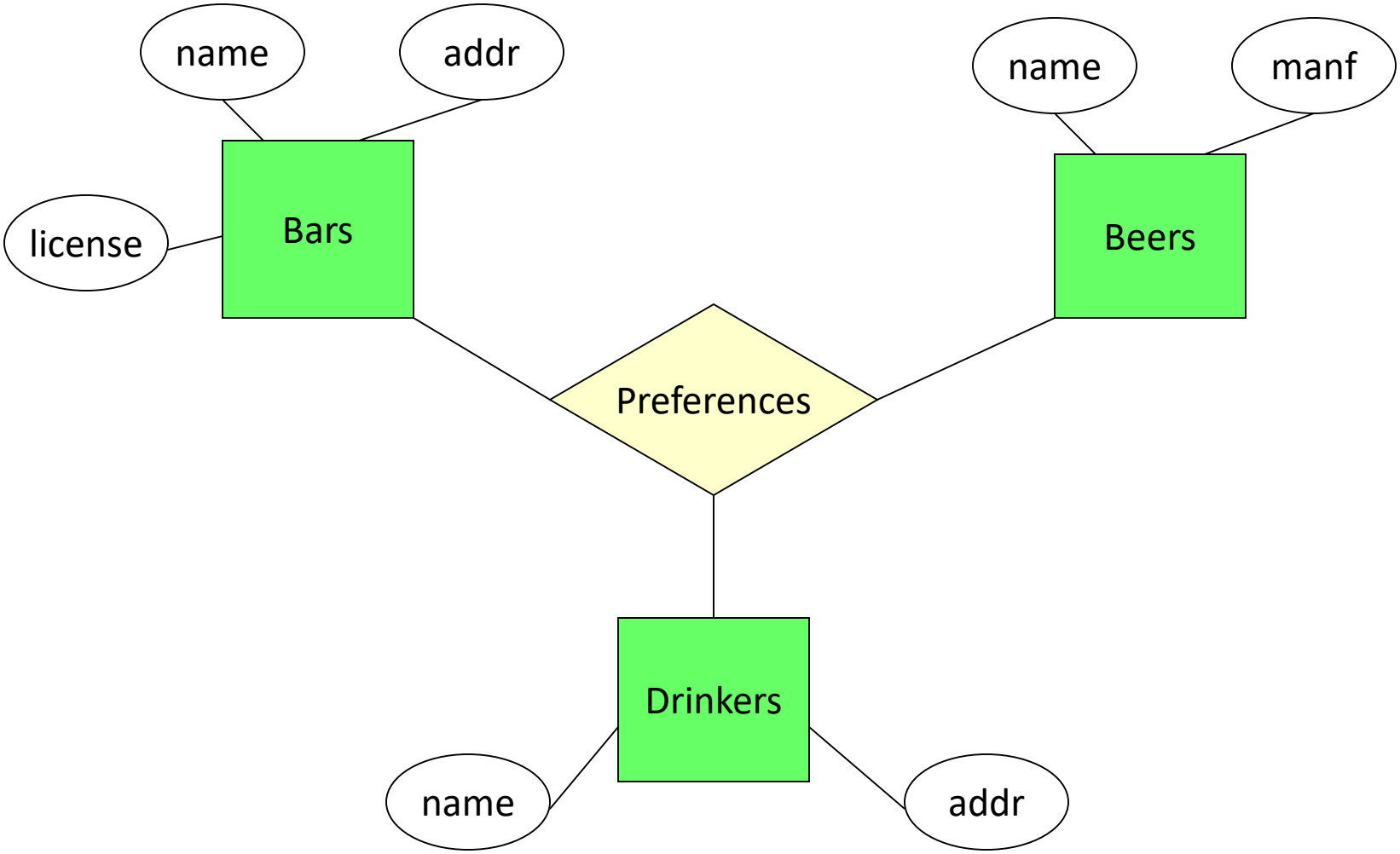




Same entity in different roles

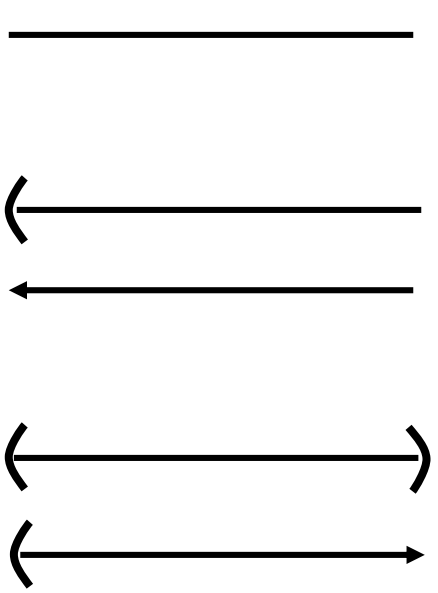


Ternary relationships



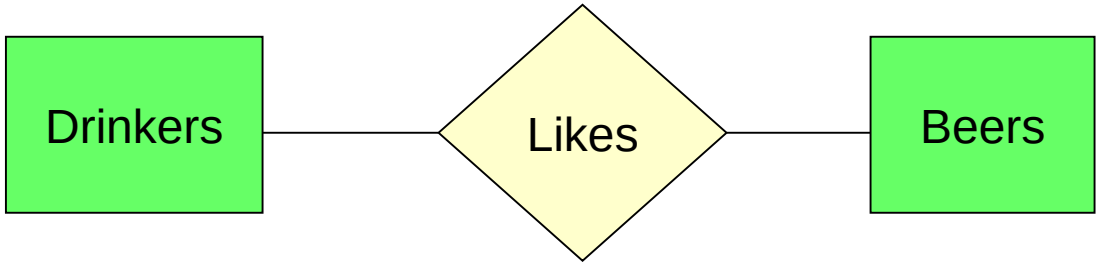
Multiplicity of relationships

- many-to-many (binary or ternary)
- one-to-many
  - mandatory:
  - optional:
- one-to-one
  - both mandatory:
  - one mandatory, one optional:

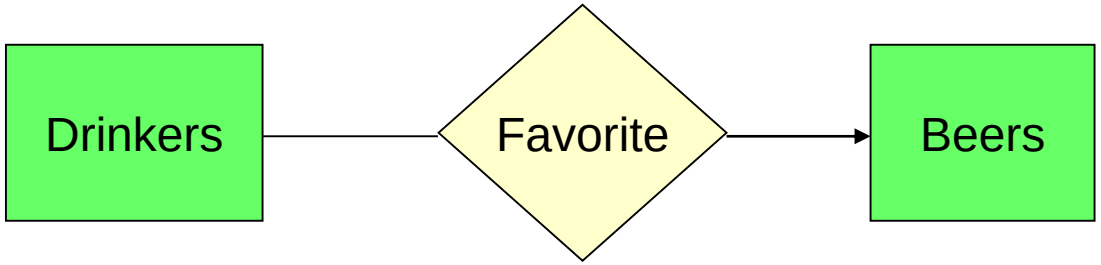


Multiplicity of Relationships

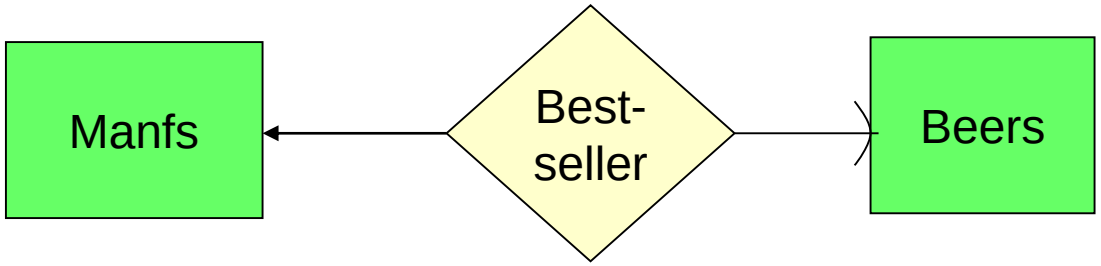
many-many



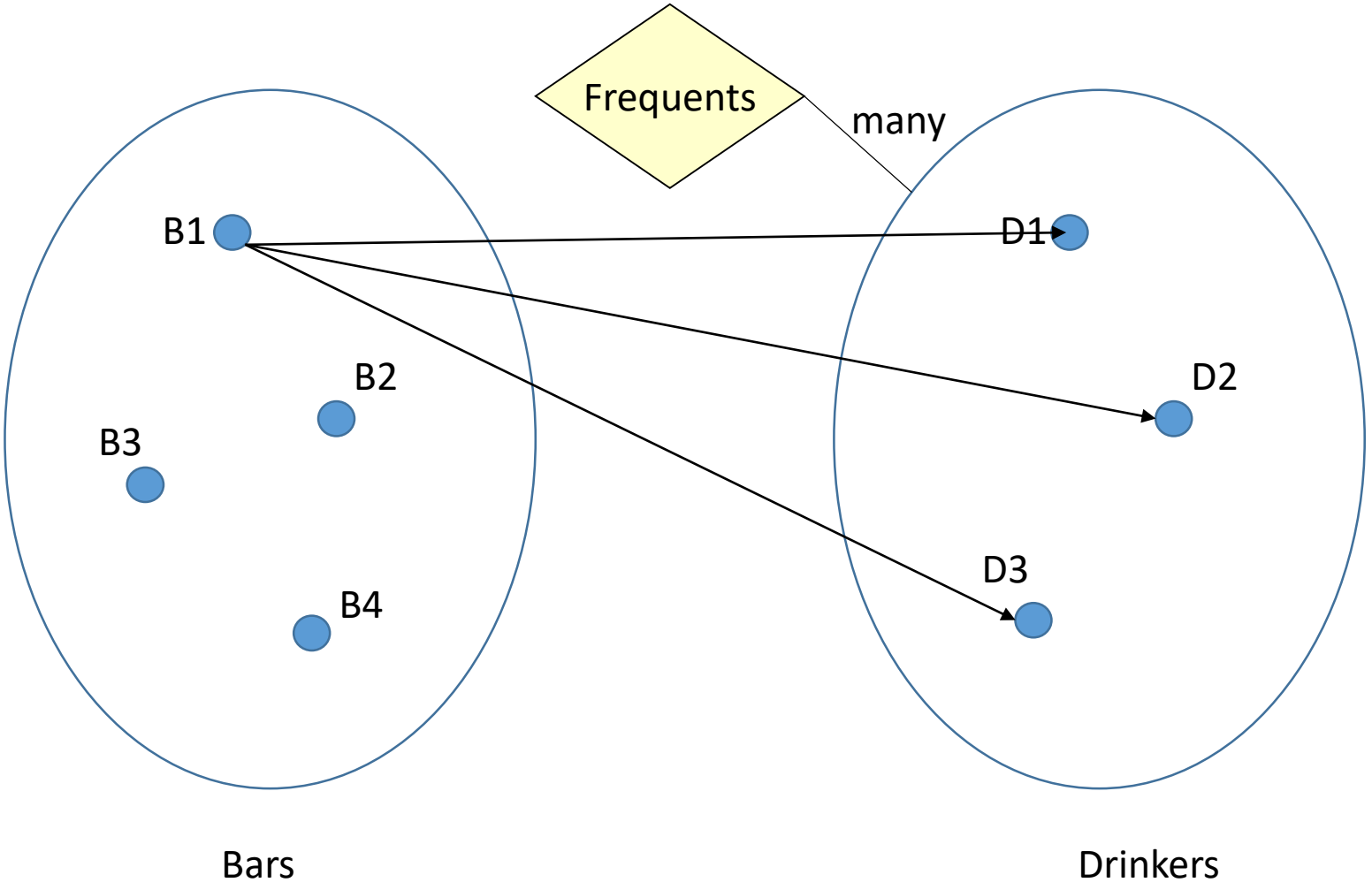
many-one



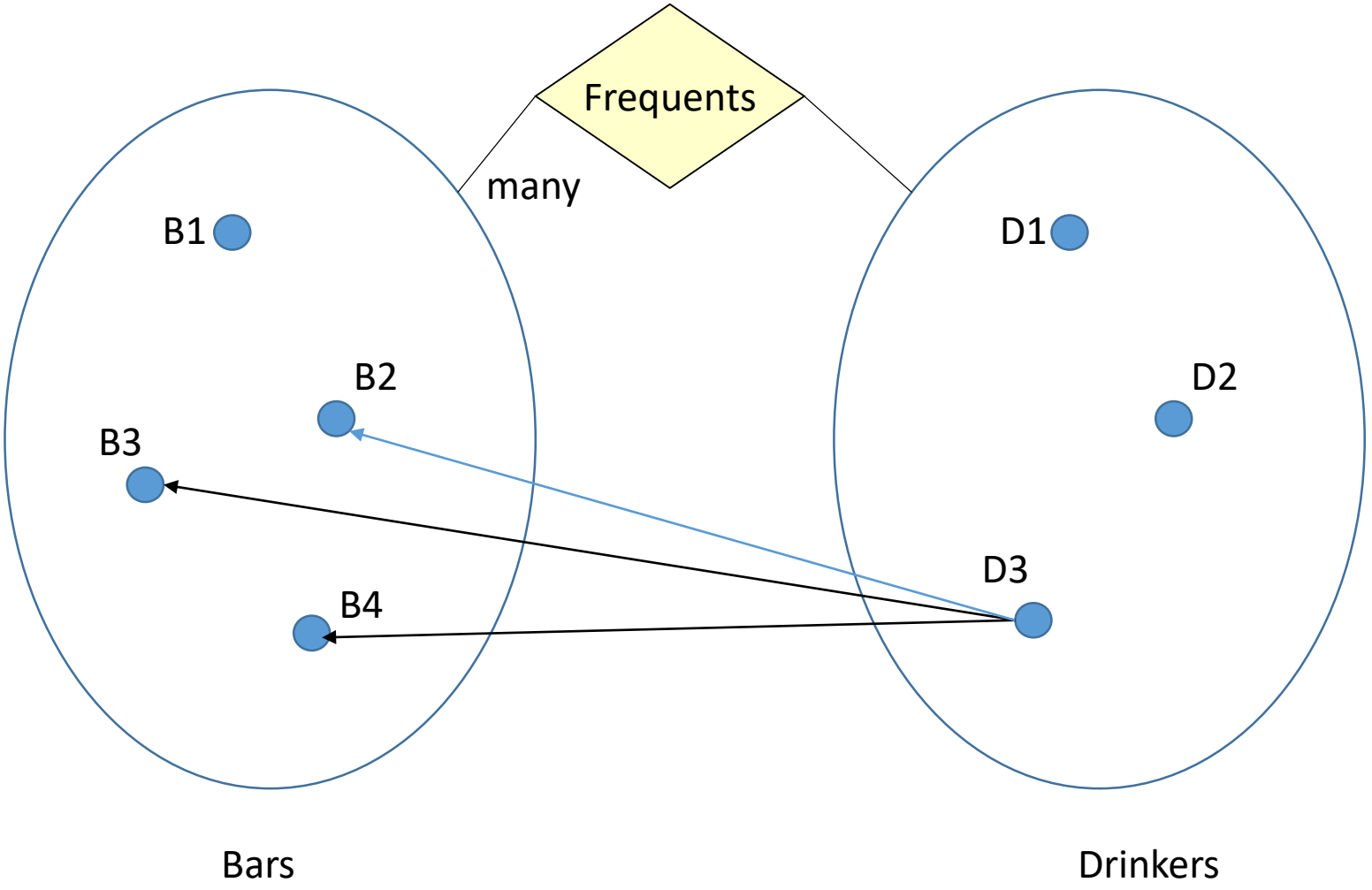
one-one



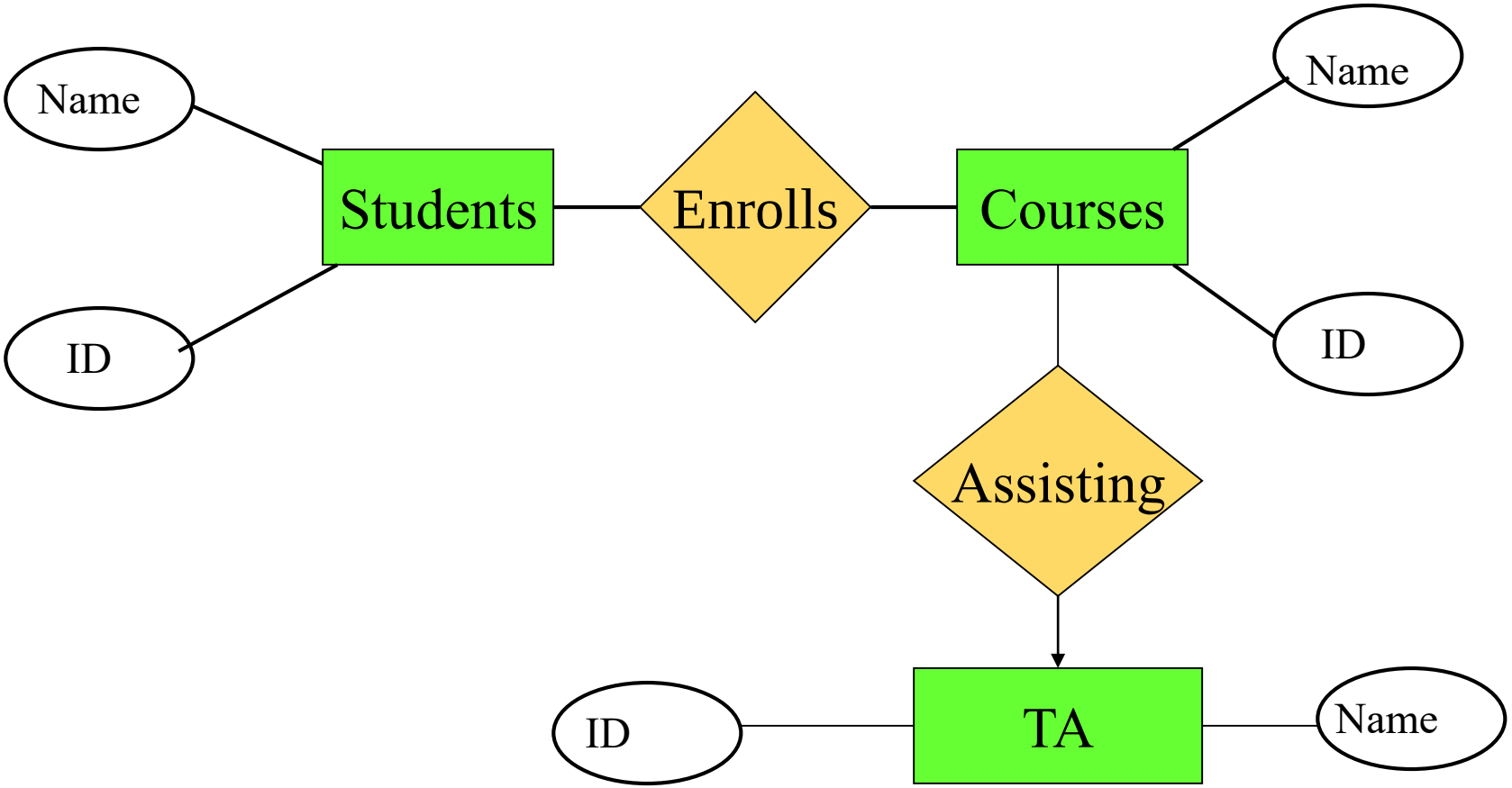
Testing multiplicities



Testing multiplicities

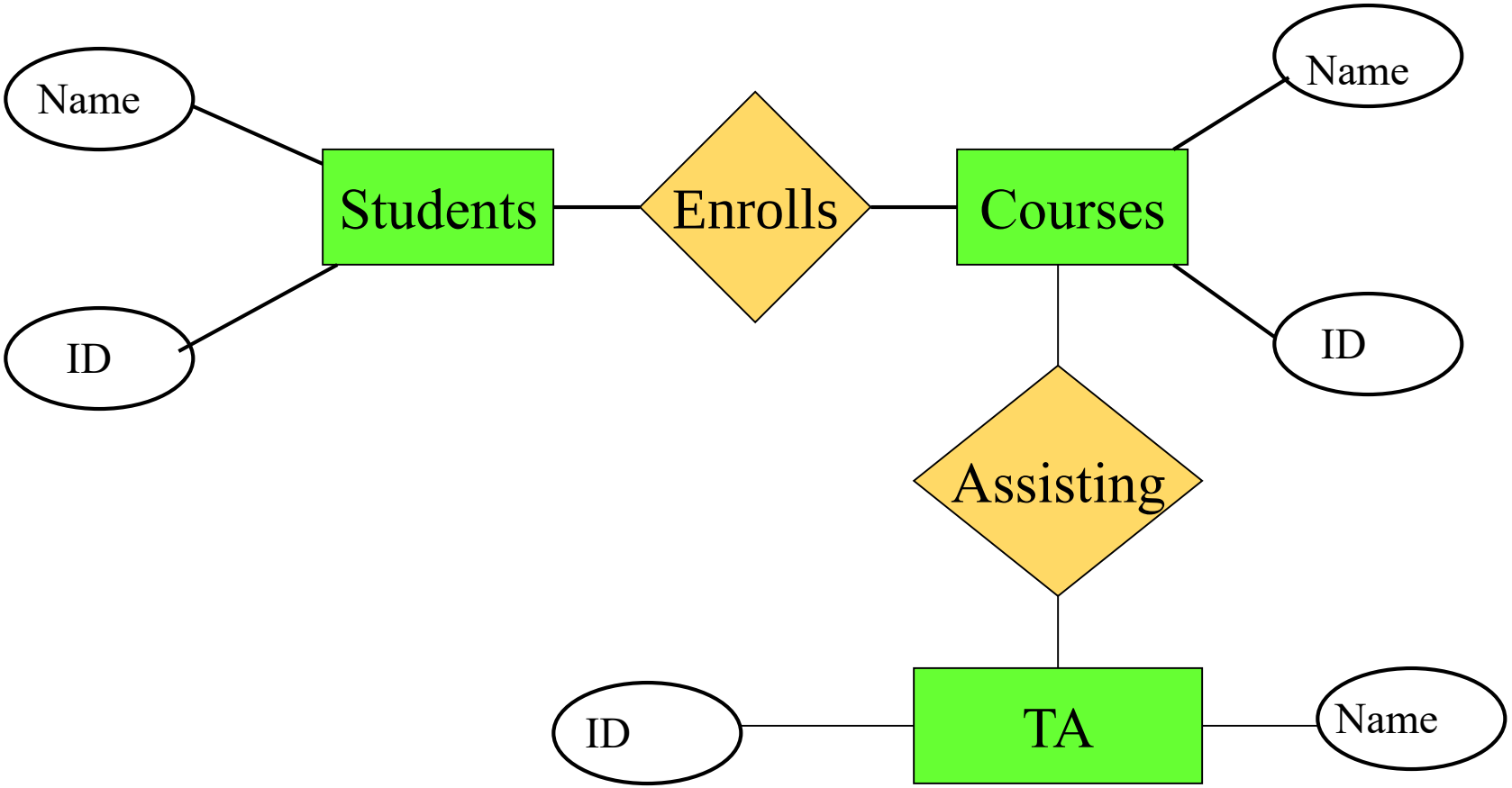


Multiplicity of multiway relationships: 1/3



At most 1 TA per course

Multiplicity of multiway relationships: 2/3



Multiple TAs per course

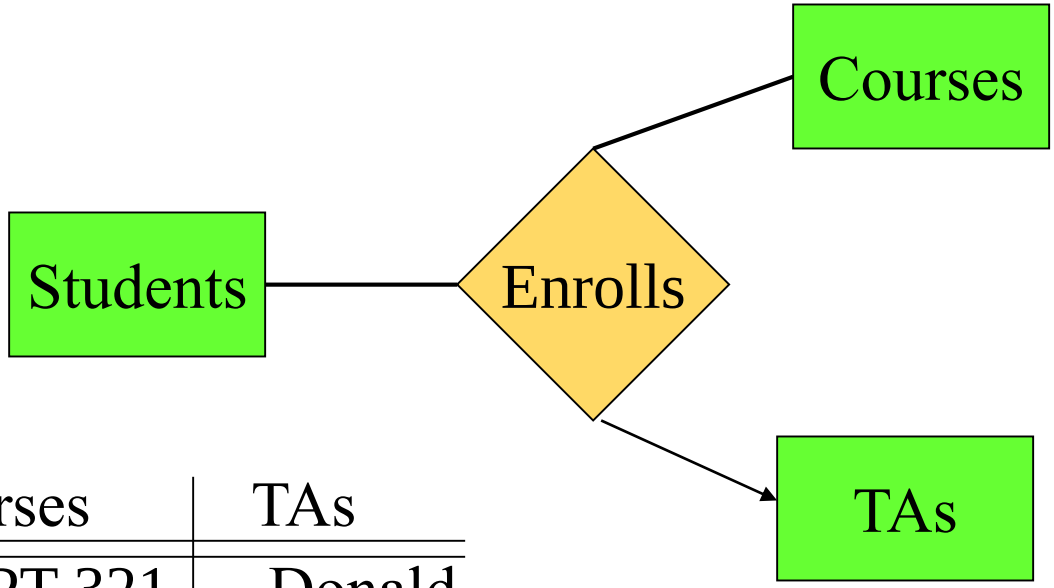
Multiplicity of multiway relationships: 2/3 problem

- Works if each TA is a TA of all students
  - Student and TA connected through Course

- But what if students were divided among multiple TAs?
  - Then a student in CMPT 321 would be related to all of the TA's for CMPT 321 —which one has helped him?

- Ternary relationship is helpful here

Multiplicity of multiway relationships: 3/3



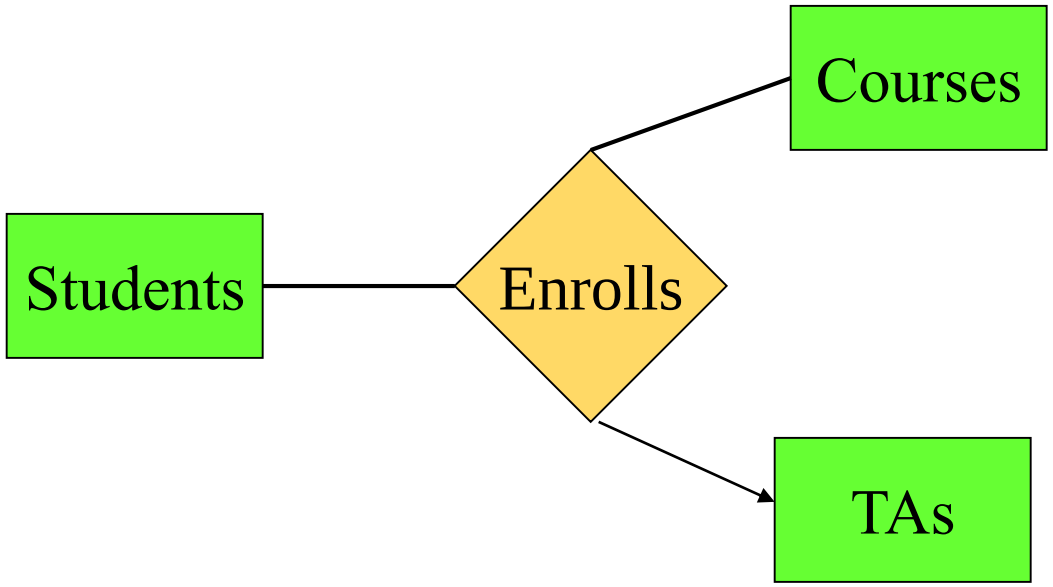
Enrolls entries:

Students	Courses	TAs
Condi	CMPT 321	Donald
George	CMPT 321	Dick
Alberto	CMPT 321	Colin
…	…	…

Enrolls determines *TA*:

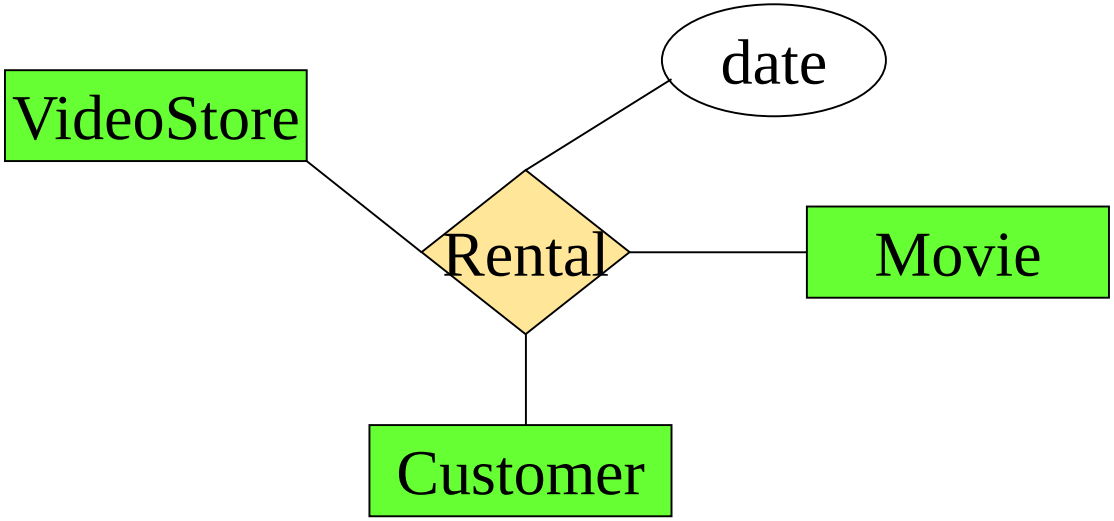
(student, course) → at most one TA

Multiplicity test for multiway relationships: example 1



Take every PAIR of entities and see to how many entities it is related in the third:
(Student A, Course B) -> 1 TA
(Course B, TA C) -> multiple students
(Student A, TA C) -> possibly multiple courses over the years

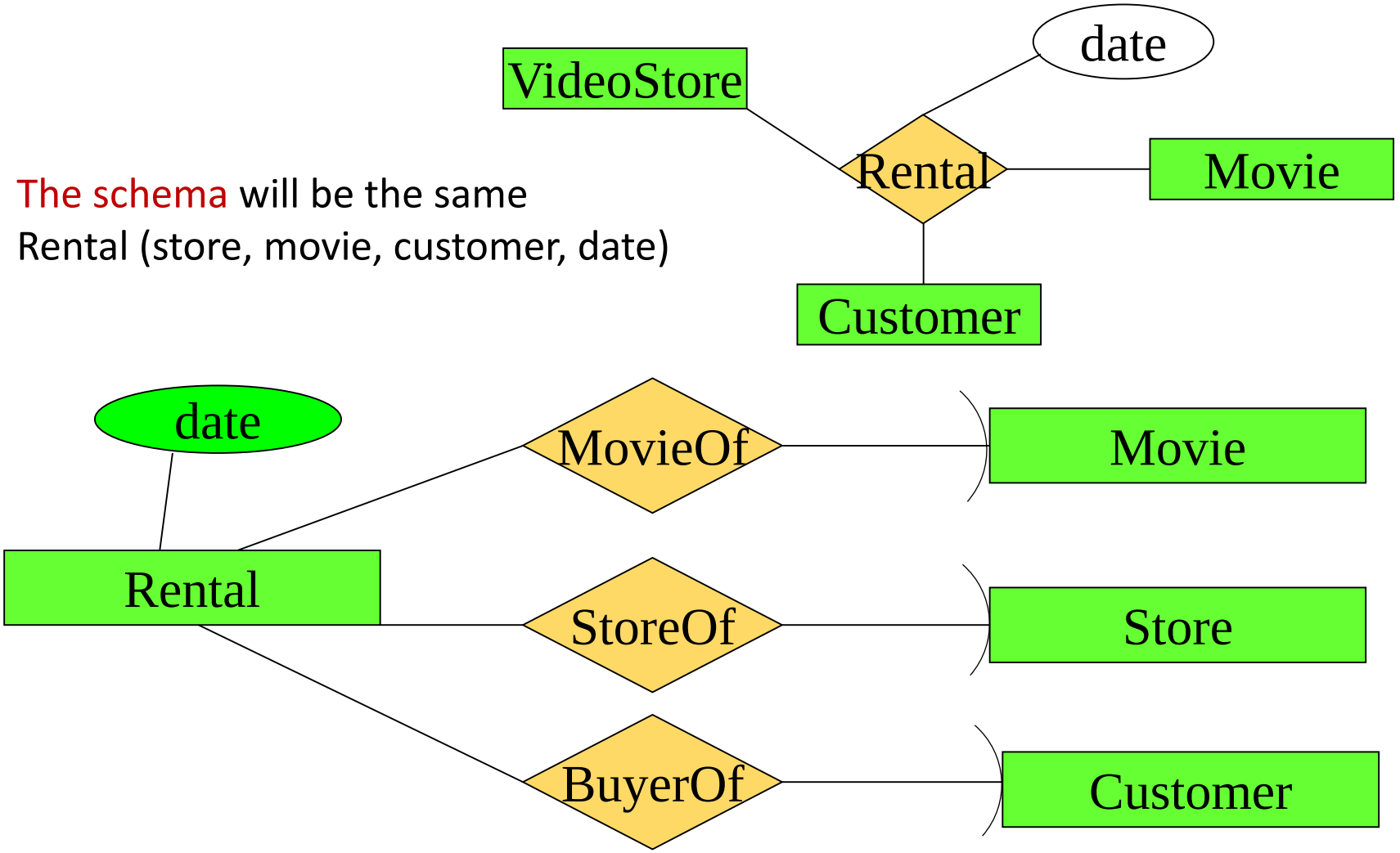
Multiplicity test for multiway relationships: example 2



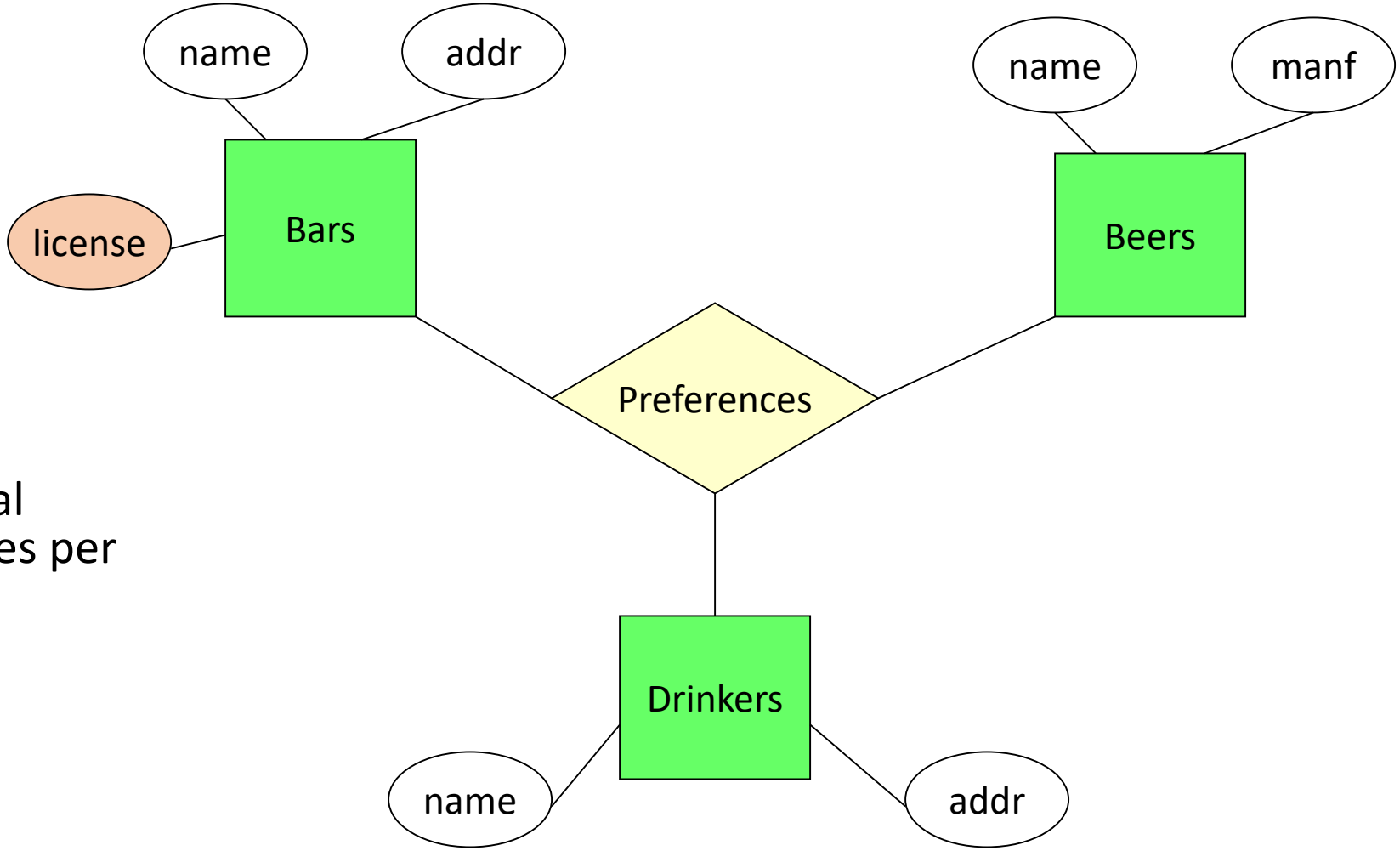
Where should we put arrows here?

Decomposing ternary relationships into binary using entity set

The schema will be the same
Rental (store, movie, customer, date)

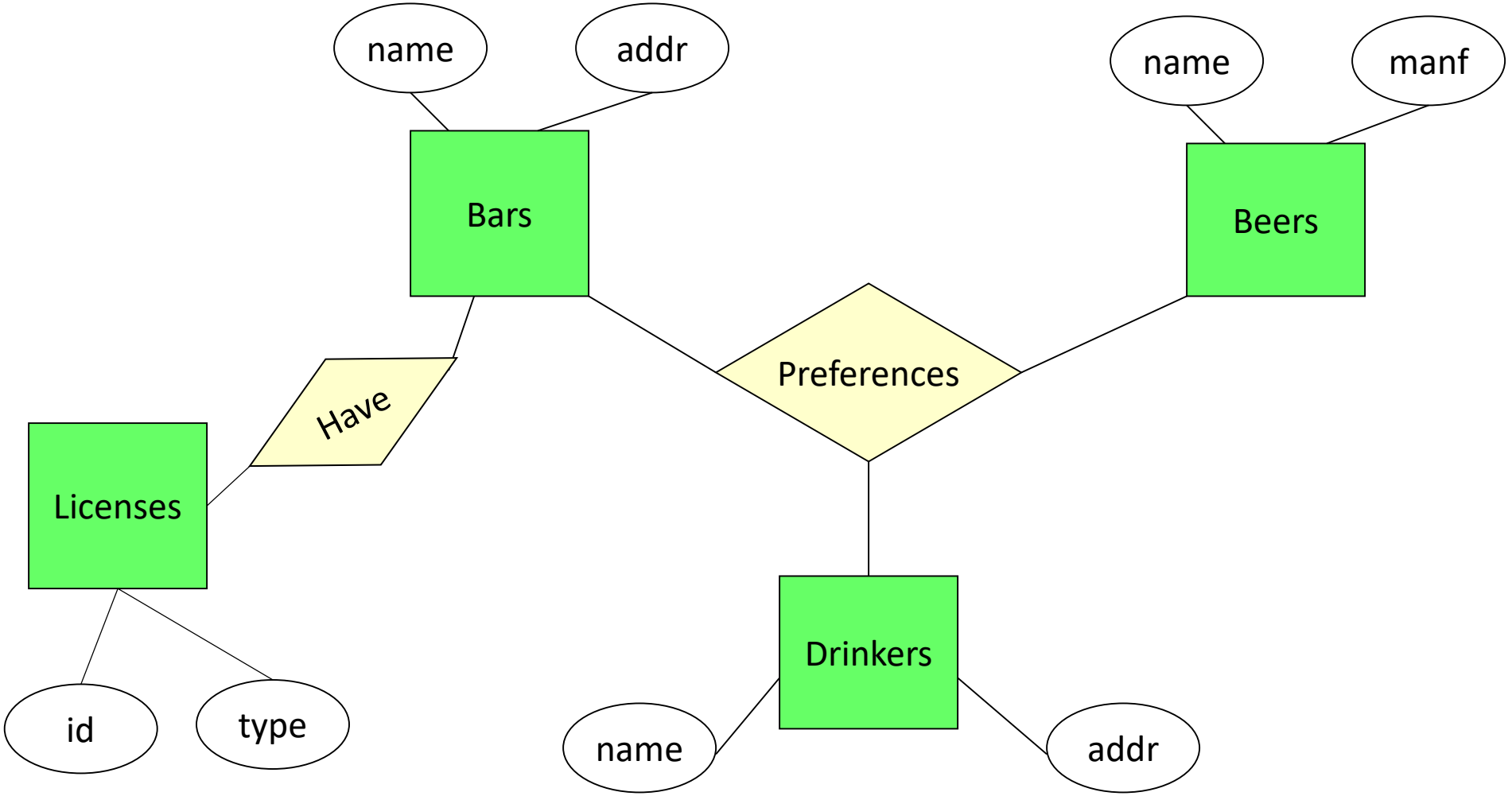


Multivalued attributes are not allowed

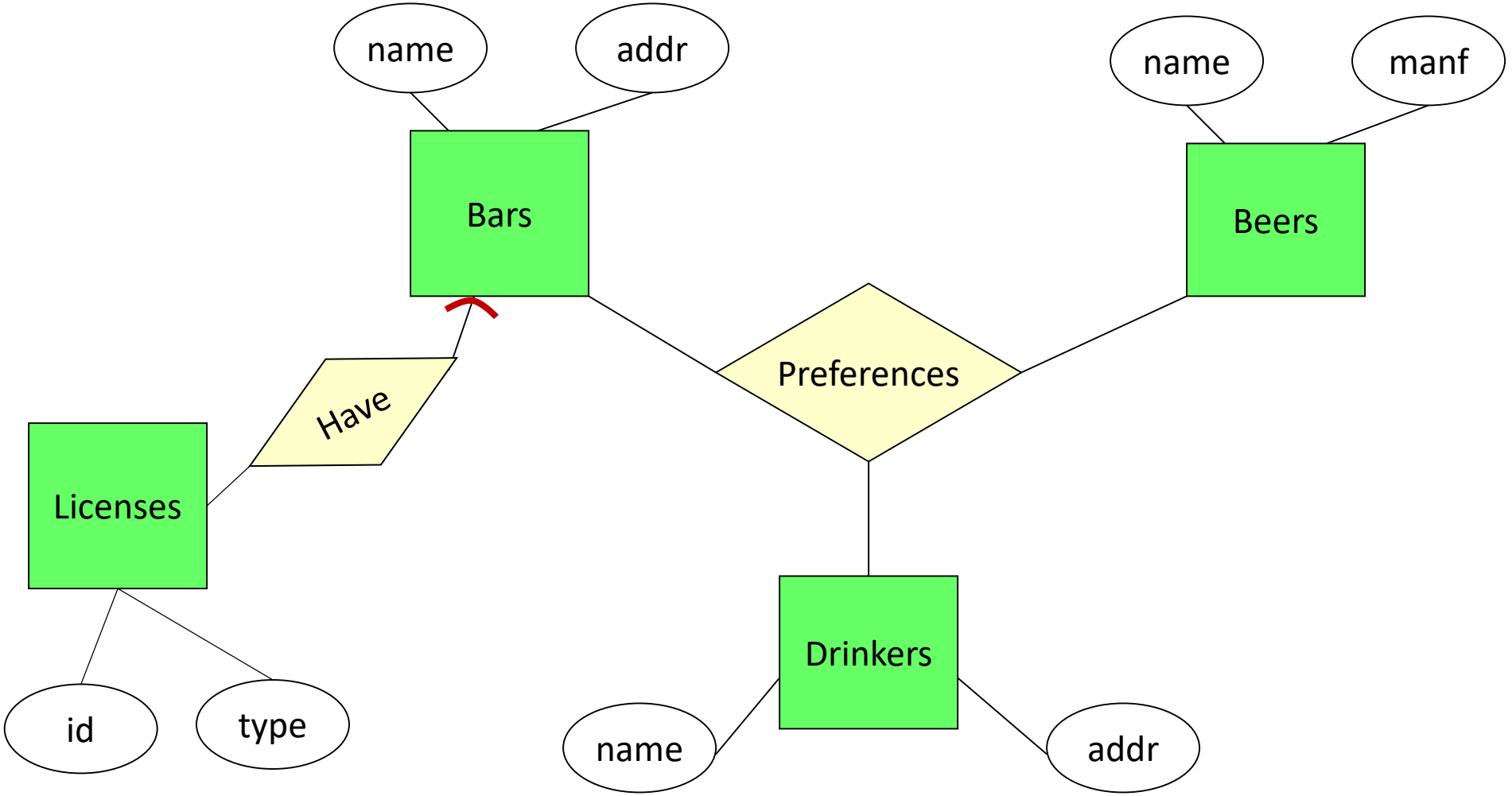


Several licenses per bar

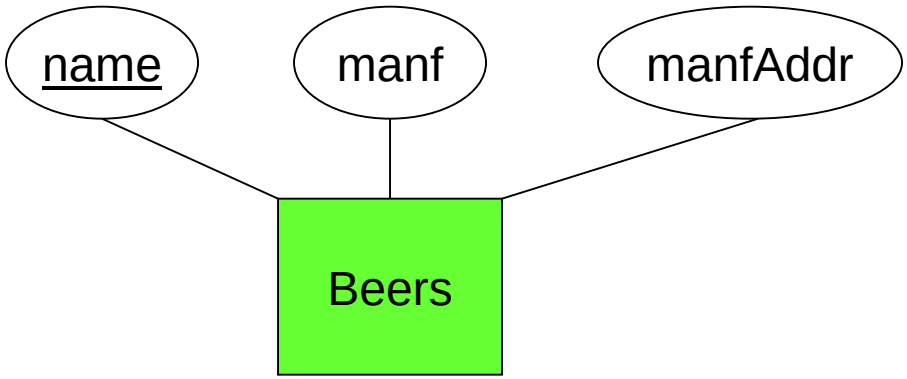
Multivalued attributes should be pulled out into an entity



Even if they form relationship with only a single entity



Entity Sets Versus Attributes



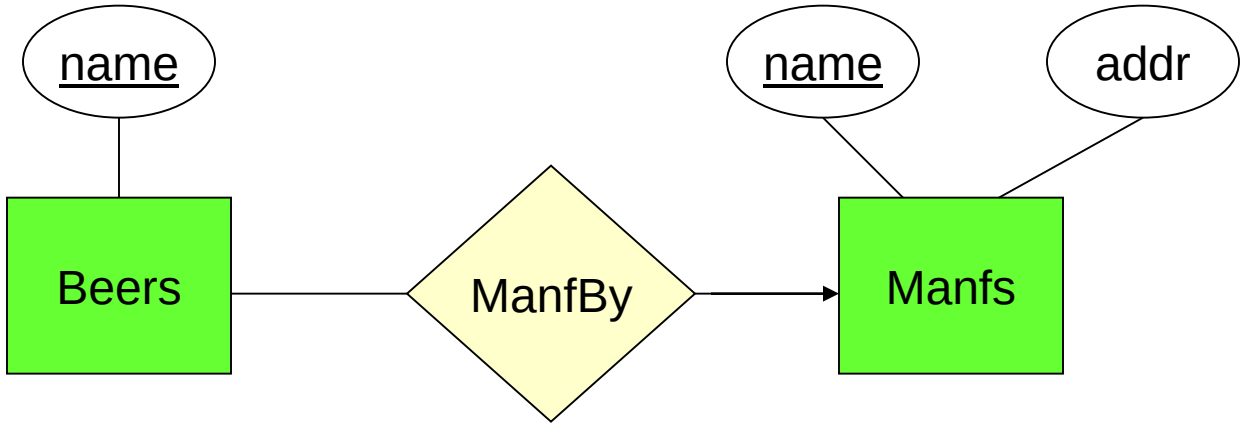
Example of bad design. Why?

1. Repeats the manufacturer's address once for each beer
2. Loses the address if there are temporarily no beers for a manufacturer

What to do in this case?

From attributes to entity sets

- *Manfs* deserves to be an entity set because of the non-key attribute *addr*
- *Beers* deserves to be an entity set because it is the "many" of the many-one relationship *ManfBy*



When to replace an attribute with an entity set

- An entity set should satisfy at least one of the following conditions:
  1. It is more than the name of something - it has at least one non-key attribute

     or

  2. It is the "many" in a many-one or many-many relationship

- Intuition
  - A "thing" in its own right => Entity Set
  - A single-valued "detail" about some other "thing" => Attribute

Basic E/R design: summary

- Identify entities (entity sets) and their attributes
- Identify relationships between entities (relationship sets) and their attributes
- Are there recursive (self)-relationships?
- Are there different roles for the same entity?
- Do we need ternary relationships?
- Attribute or entity?
- Mark multiplicity

Entity-Relationship diagrams: refinements

Lecture 01.02

By *Marina Barsky*

1. Keys
2. Subclasses
3. Week entity sets
4. Aggregates

Keys

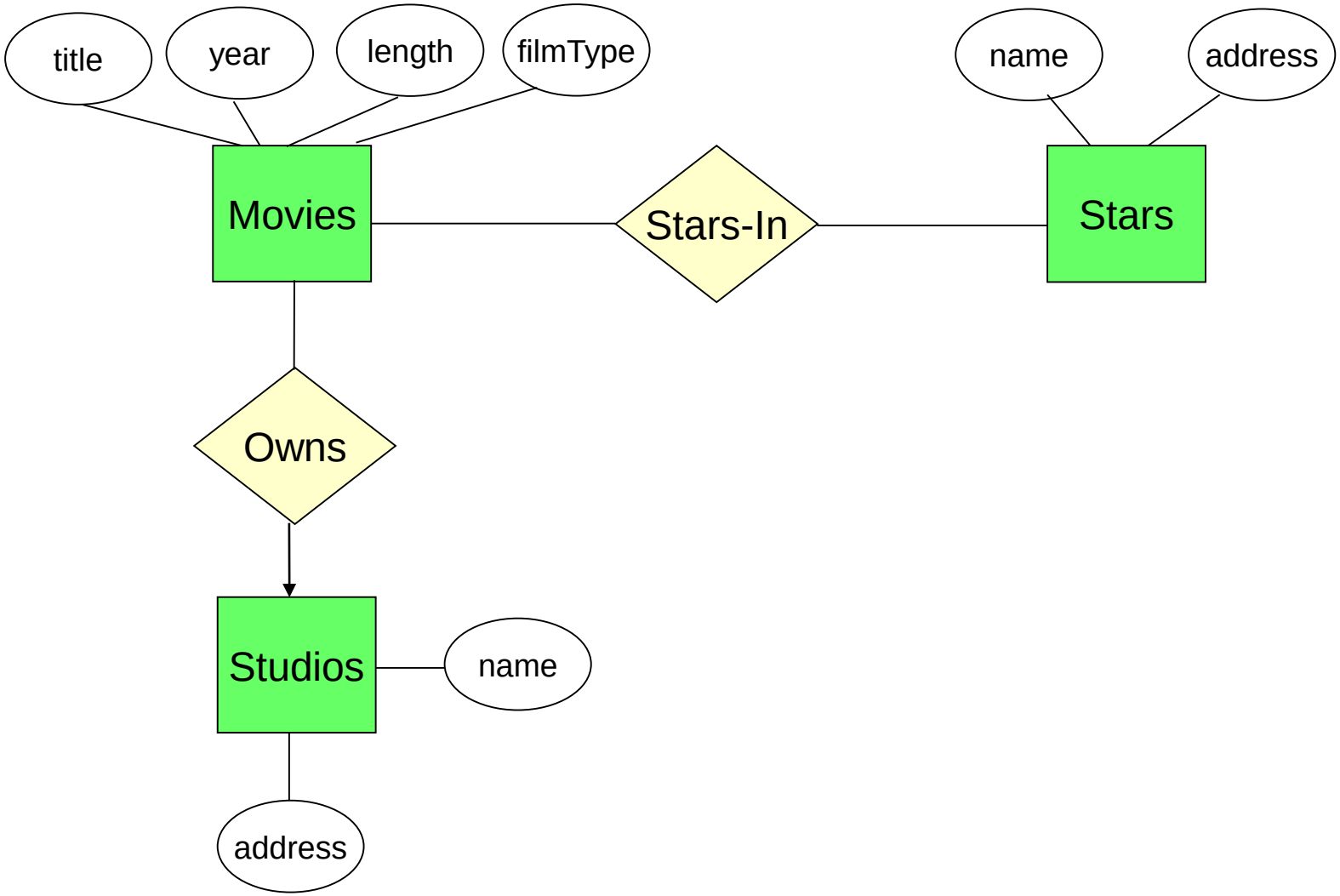
for entity sets

Keys

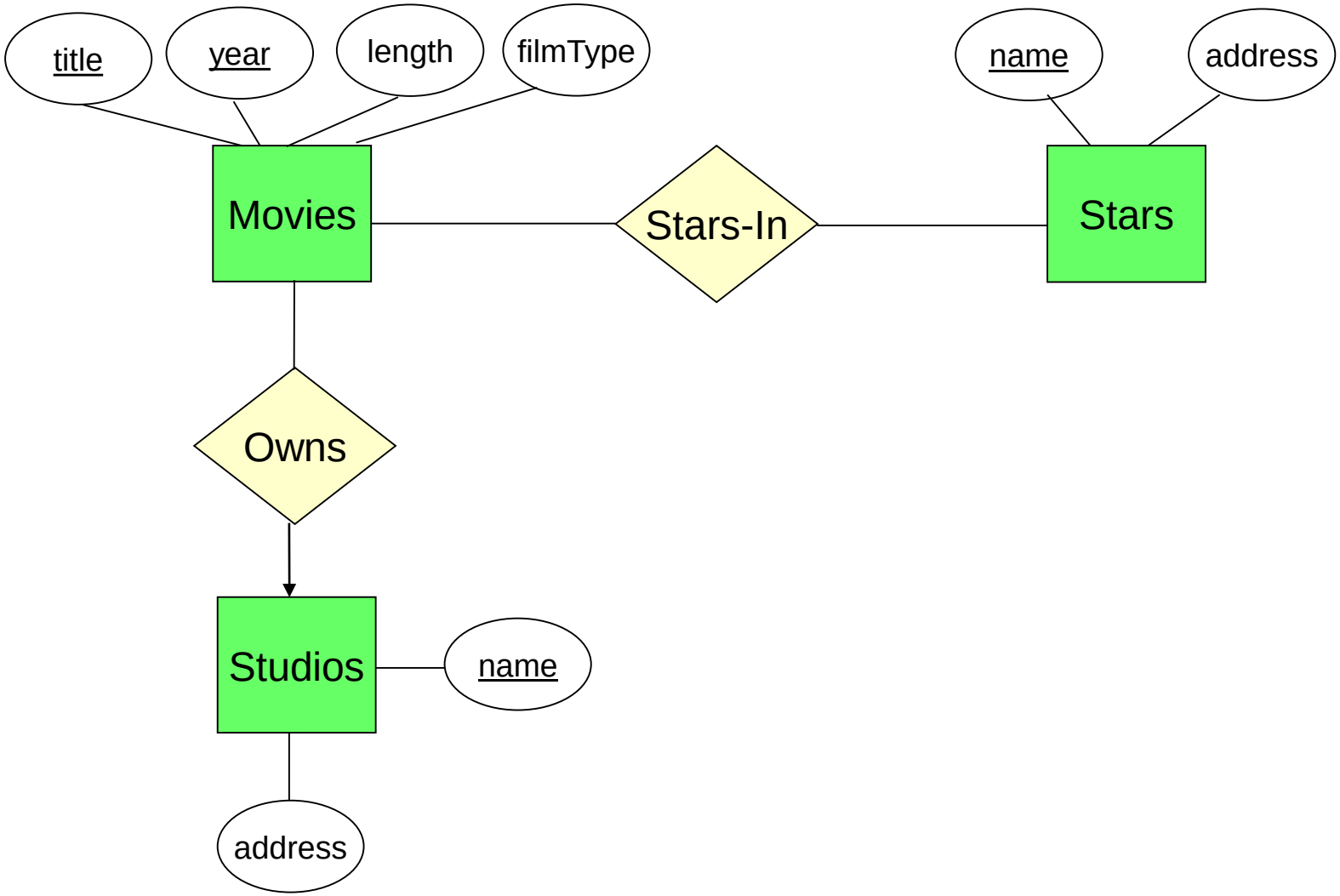
- A ***key*** (for an entity set) is a set of attributes such that **no two entities agree on all the attributes of the key**

- In E/R, we underline the key attribute(s)

Keys?


title
year
length
filmType
name
address
Movies
Stars-In
Stars
Owns
Studios
name
address

Keys


title
year
length
filmType
name
address
Movies
Stars-In
Stars
Owns
Studios
name
address

Internal vs. surrogate Keys

- In most cases, a key is formed by one or more attributes of the entity itself (***internal*** keys)
- Often, people introduce attributes whose role is to serve as a ***surrogate*** key:
  - Companies assign employee ID's to all employees, and these ID's are carefully chosen to be unique numbers.
  - In US/Canada everyone has a SSN/SIN
  - Students ID's in universities
  - Driver license numbers
  - Automobile registration numbers

Rules about key selection

- Attributes with possible missing values cannot form a key
- Internal keys preferable to surrogate key
- One/few attributes is preferable to many attributes

Possible problems with internal keys: multiple attributes and strings

- **Wasteful**
  - e.g. Movies (title, year,…): 2 attributes, ~16 bytes
  - Number Of movies ever made << 2^{32} (4 bytes) => Integer movieID key saves 75% space
- **Break encapsulation**
  - e.g. Parent (firstName, lastName, phone,…)
  - Security/privacy hole => Integer parentID prevents information leaks
- **Can change**
  - Name or phone number change?
  - Parent and child with same name?
  - Parent with no phone?

If we have a global authority over our database – we create a surrogate key

Numeric surrogate IDs are always available, immutable, unique

Also: computers are really good at integers

Inheritance

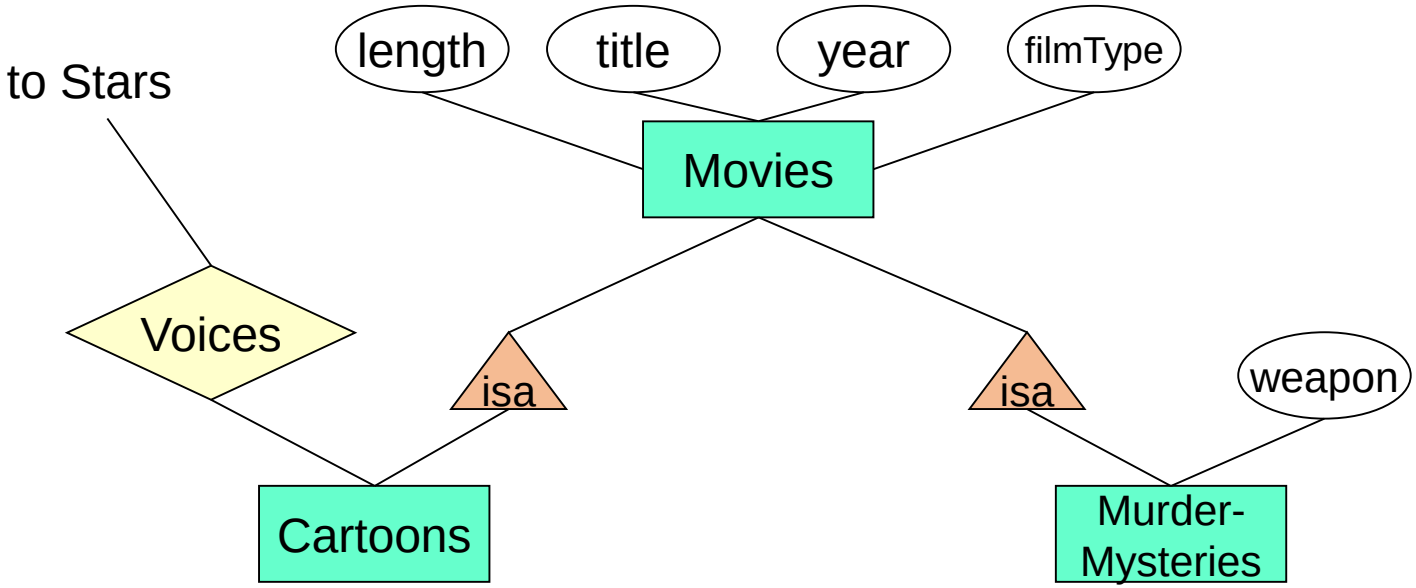
in the ER model

Subclasses

- Sometimes, an entity set contains certain entities that have special properties not associated with all members of this entity set.
- In this case it is useful to define special-case entity sets, or *subclasses*, each with its own attributes and relationships

Subclasses

Relate parent with child by a special (1-1) relationship called **isa**



Inheritance in the E/R Model

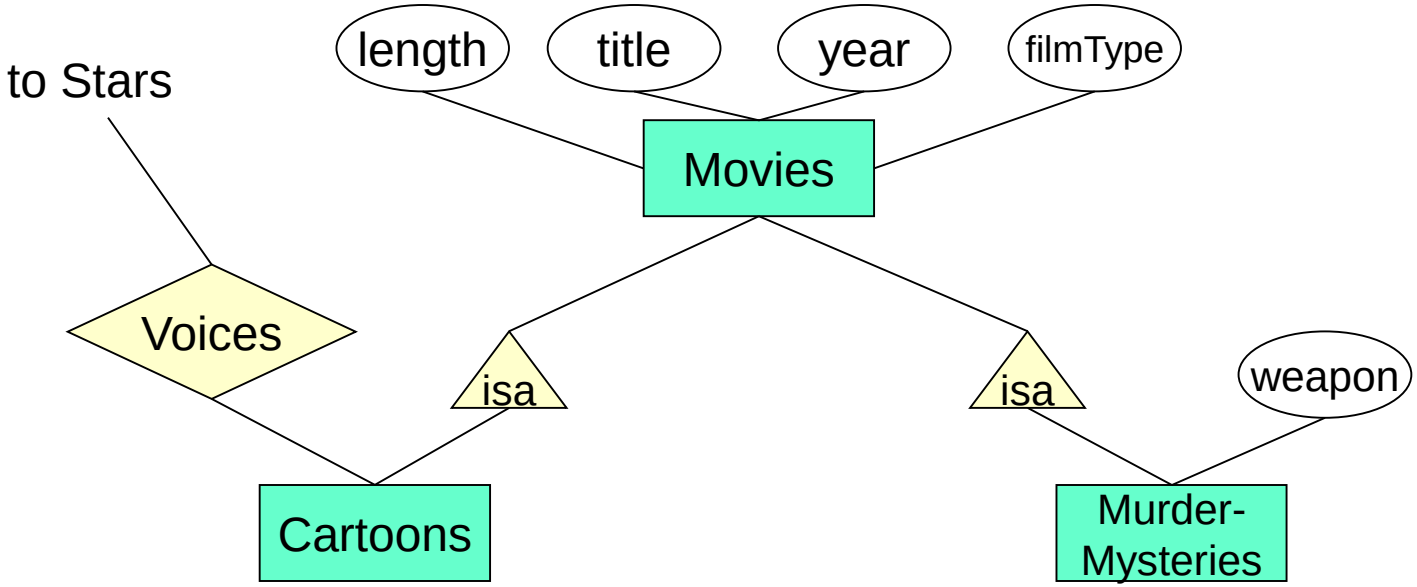
- In the object-oriented world, property values are stored in one place only:

  Subclasses inherit the property from superclasses

- In contrast, E/R entities participate in all subclasses to which they belong

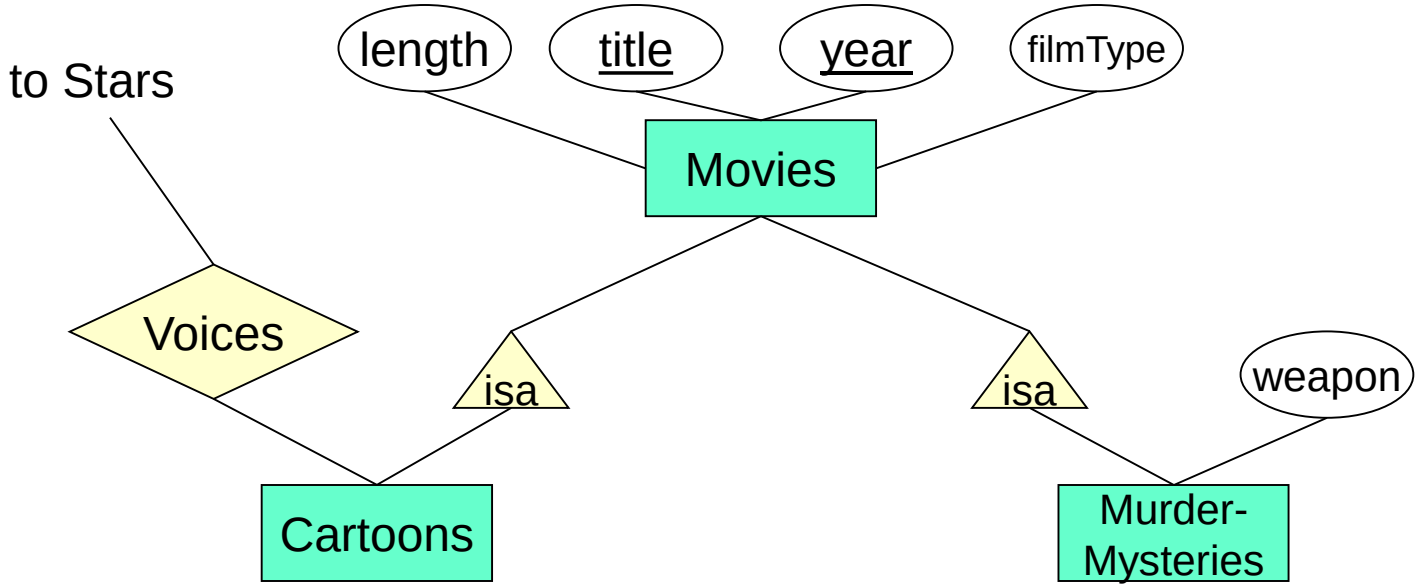
Example

- **Roger Rabbit**, which is both a **cartoon** and **murder-mystery**
  - will have one tuple in each of all three entity sets: **Movies**, **Cartoons**, and **Murder-Mysteries**



Keys for entity set hierarchies

In **entity set hierarchies** the key at root is **key** for all.
{title,year} is the key for **Movies**, **Cartoons** and **Murder-Mysteries**.



Weak entity sets

Weak entity sets

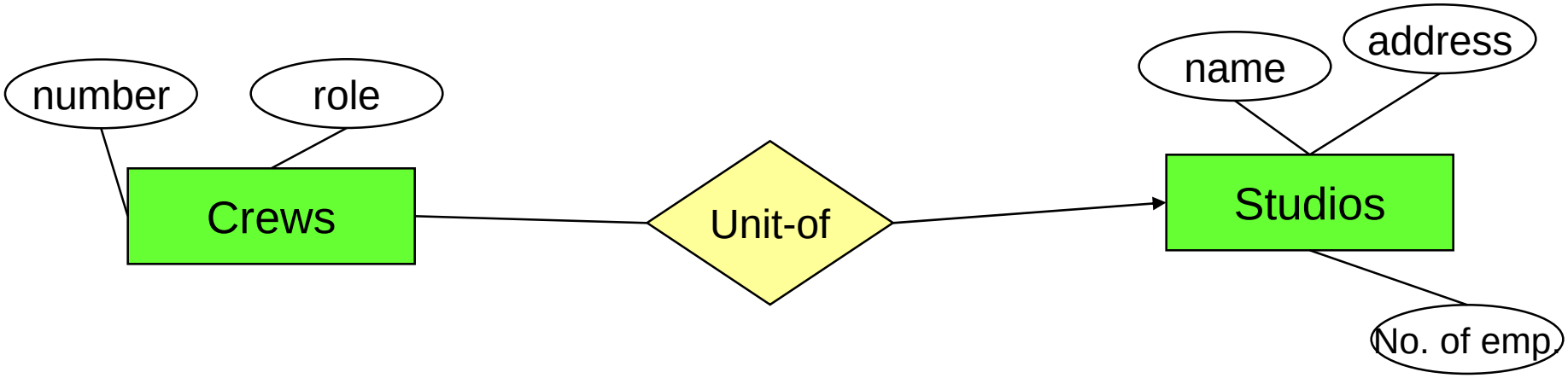
- It is possible that the key of an entity set is composed of attributes, some or all of which do not belong to this entity set
- Such an entity set is called a ***weak entity set***

- We use weak entity sets to identify sub-units of the main entity, rather than ~~sub-classes~~

Supporting relationships

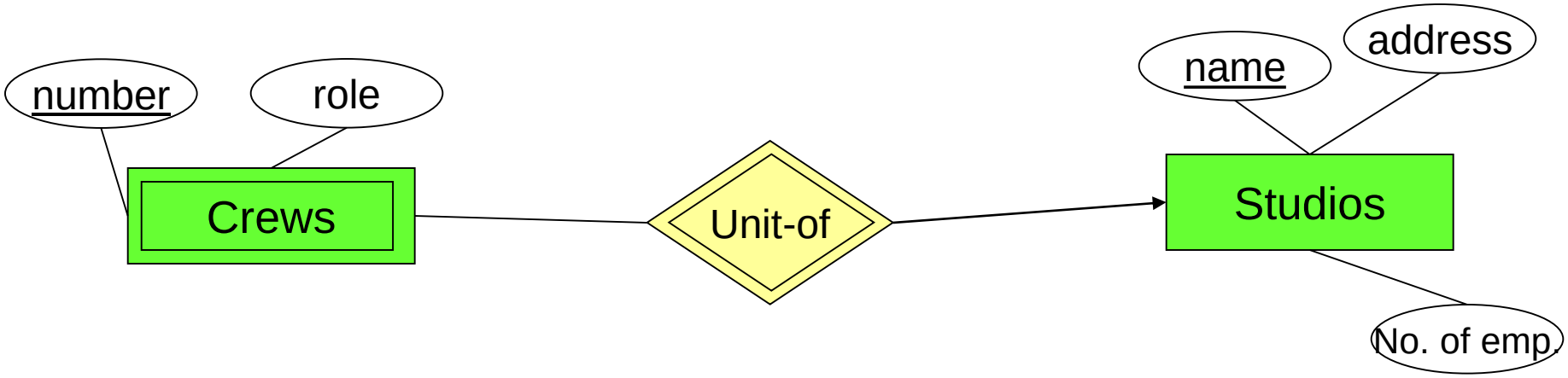
- In a weak entity set *E* the key consists of:
  - Zero or more its own attributes
  - Keys from other entities reached by many-one relationship from E
- These relationships are called **supporting relationships**

Example of a weak entity set



- E.g. "Crew 1, Special Effects" for Paramount, "Crew 1, Special Effects" for Fox etc.
- Need to add the key of **Studios**, in order to uniquely identify a crew
- **Crews** is a weak entity set

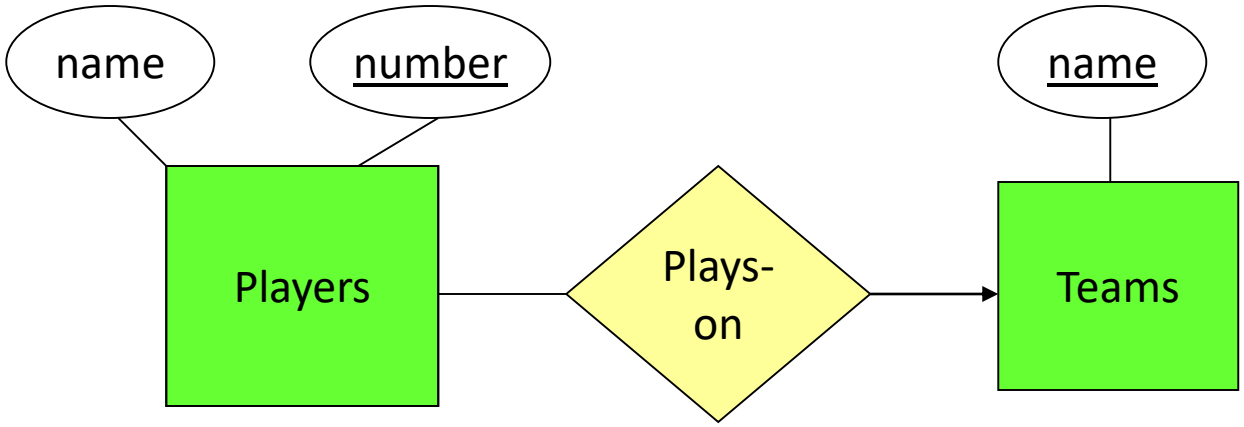
In E/R diagrams:



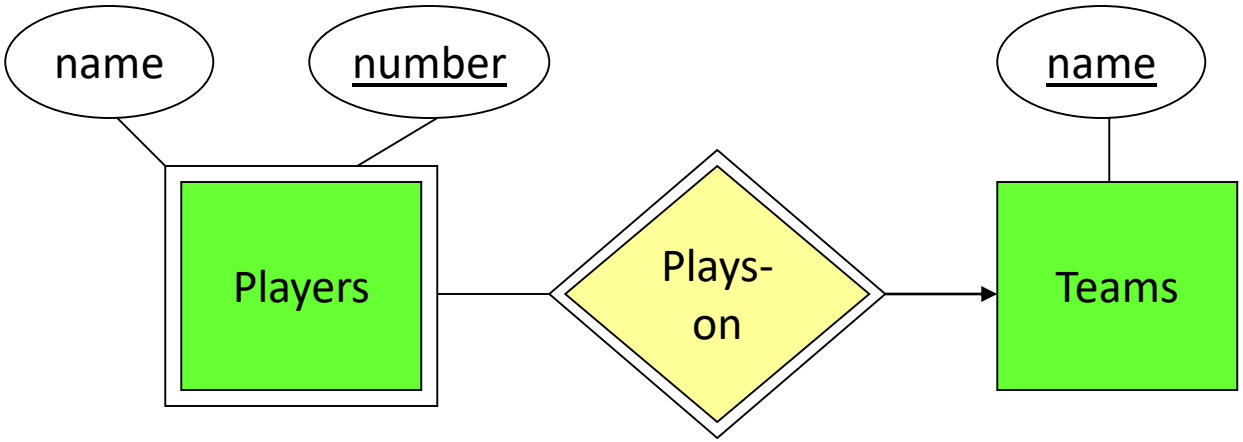
- Double rectangle for the weak entity set
- Double diamond for a *supporting* many-one relationship

Another Example – Football Players

- name is almost a key for football players, but there might be two with the same name.
- number is certainly not a key, since players on two teams could have the same number.
- But number, together with the team name related to the team by Plays-on should be unique.

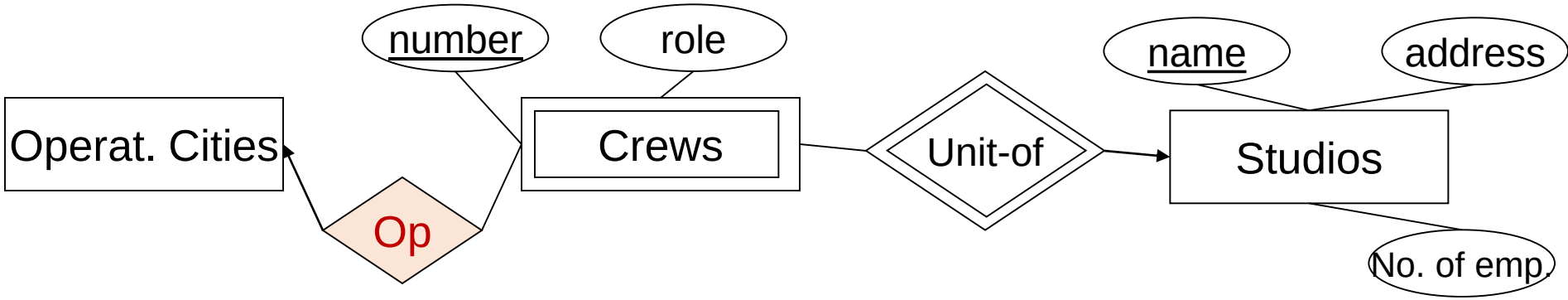


Football players - solution



Supporting vs. regular relationships

Not all the many-one relationships connecting a weak entity set to other entity sets are supporting relationships:



Weak Entity Sets – when do we need them?

- Usual reason: no global authority capable of creating unique ID's (surrogate key)

E.g.: Unlikely there could be an agreement to assign unique player numbers across all football teams in the world

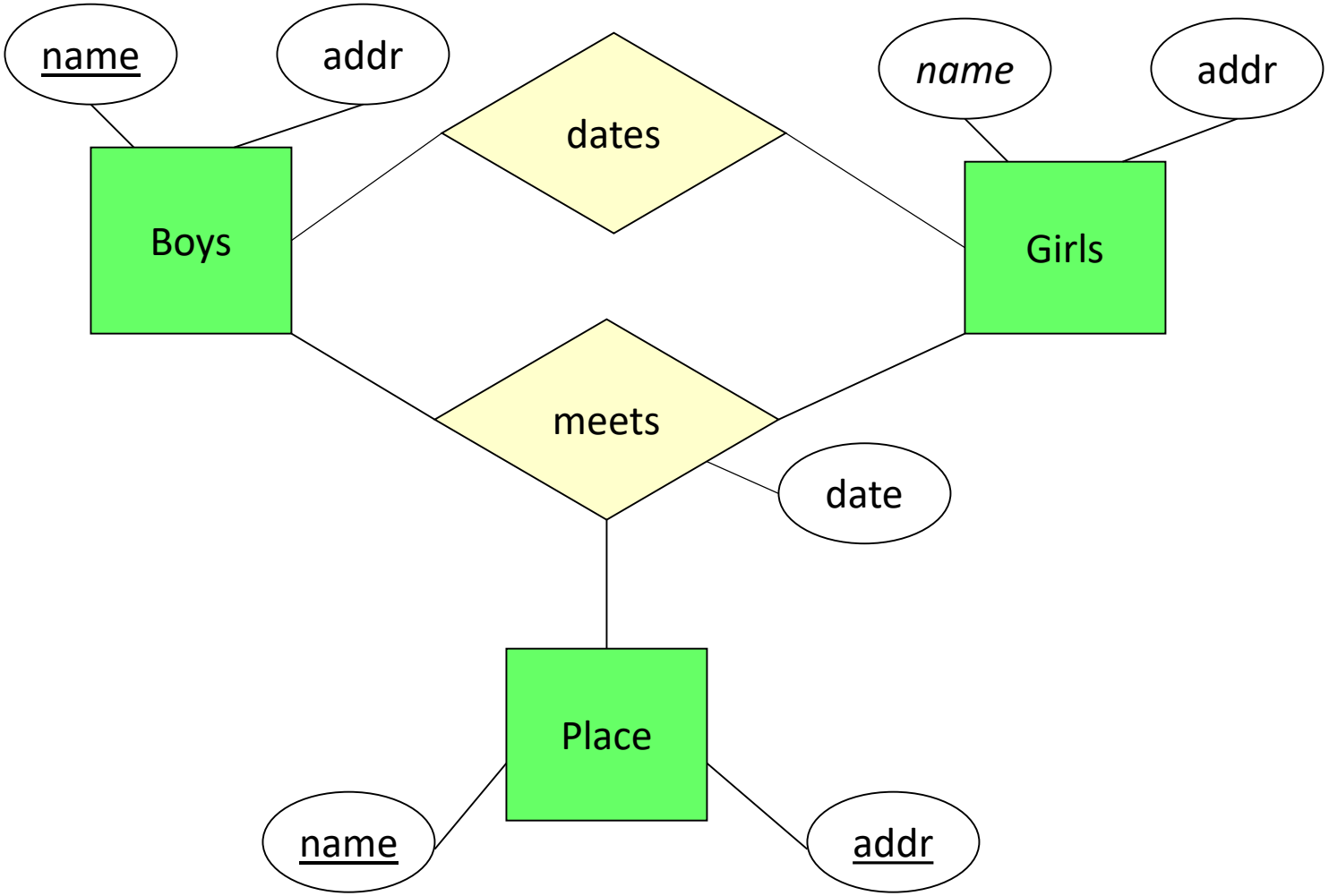
Weak Entity Sets - when we don't need them

- Beginning database designers often doubt that anything could be a key by itself
- They make all entity sets weak, supported by all other entity sets to which they are linked

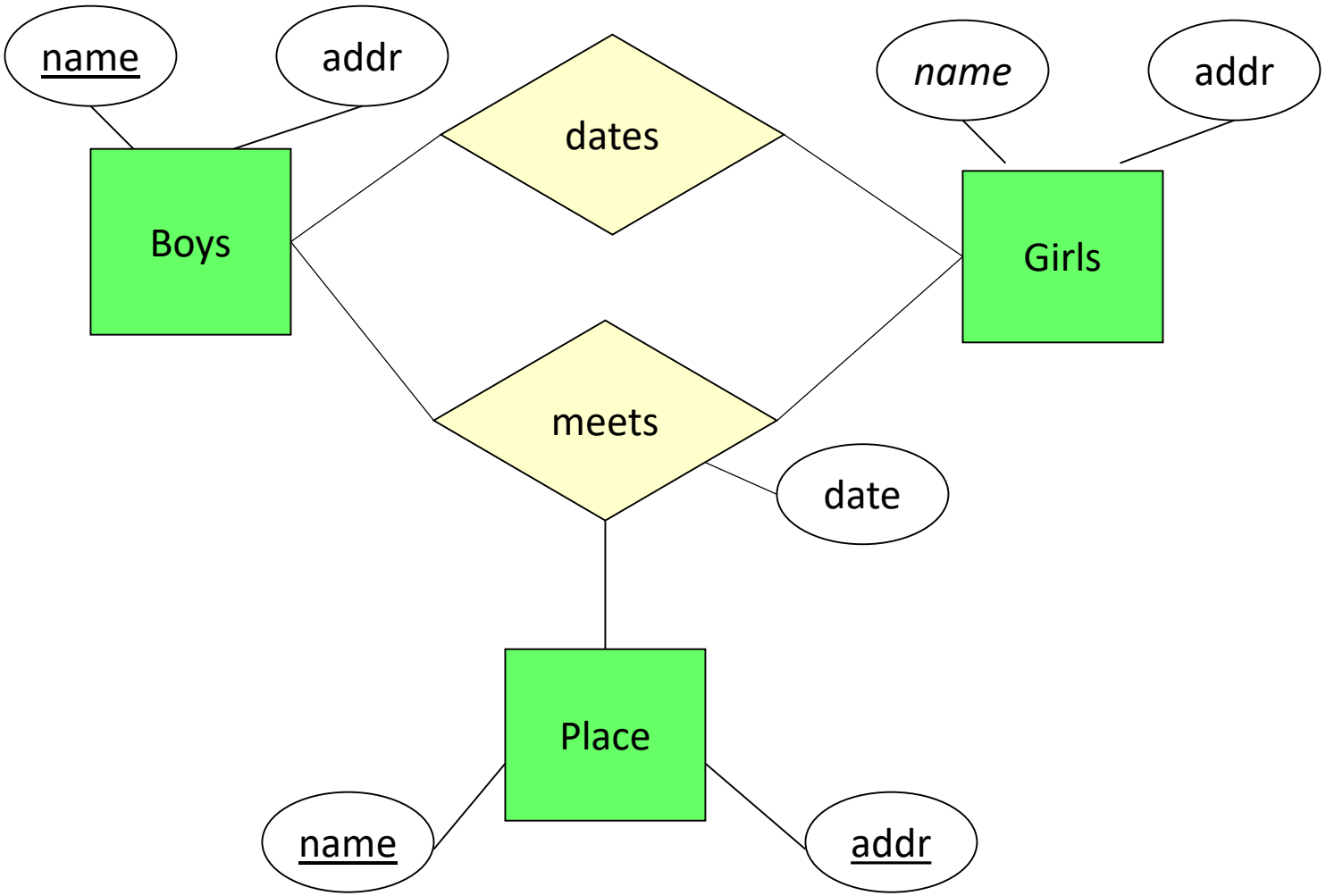
- It is usually better to create unique surrogate or use existing IDs
  - Social security number
  - Automobile VIN
  - Employee ID

Aggregation

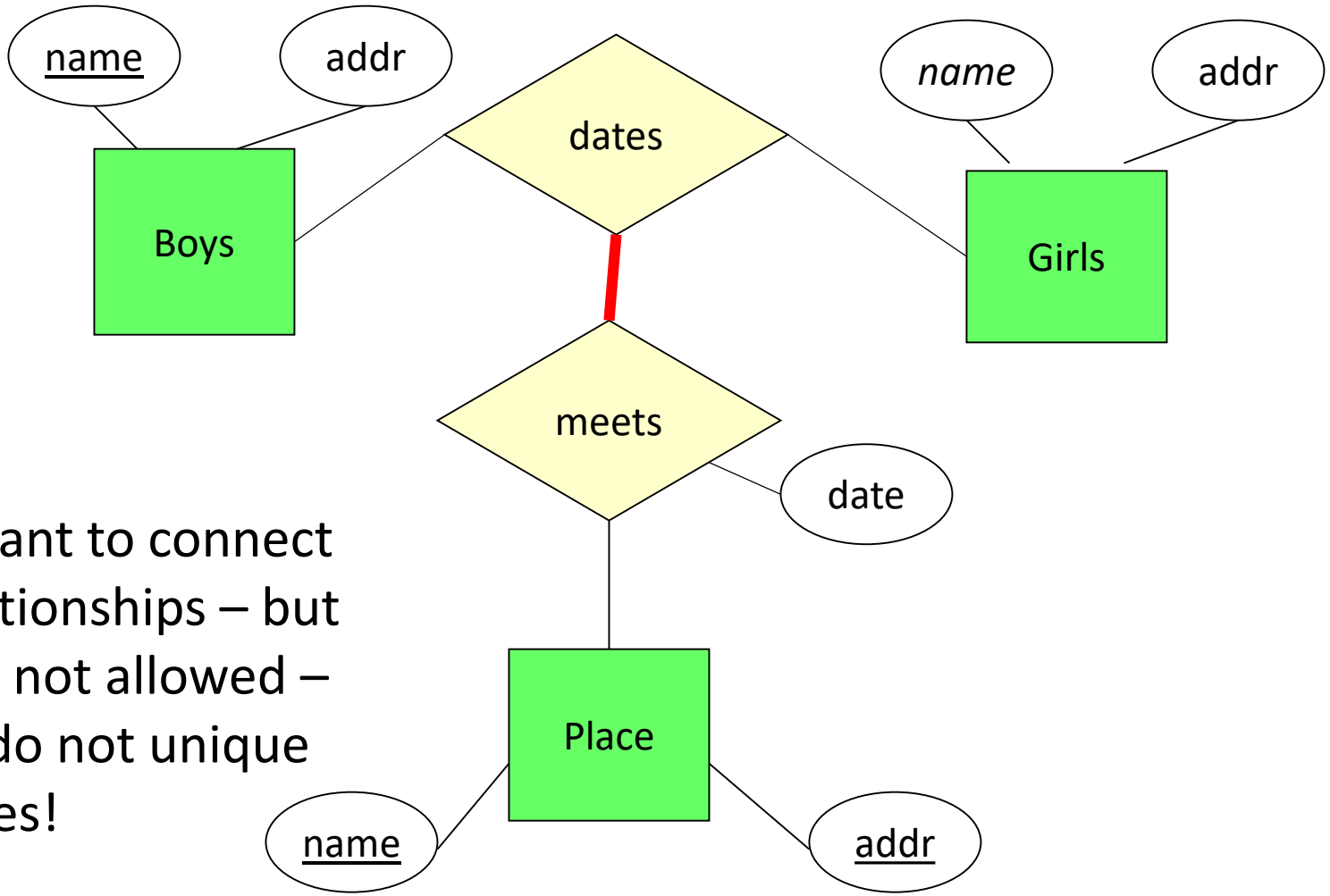
Example: redundant information in relationships



Dates relationship already defines the pair ids in *meets*

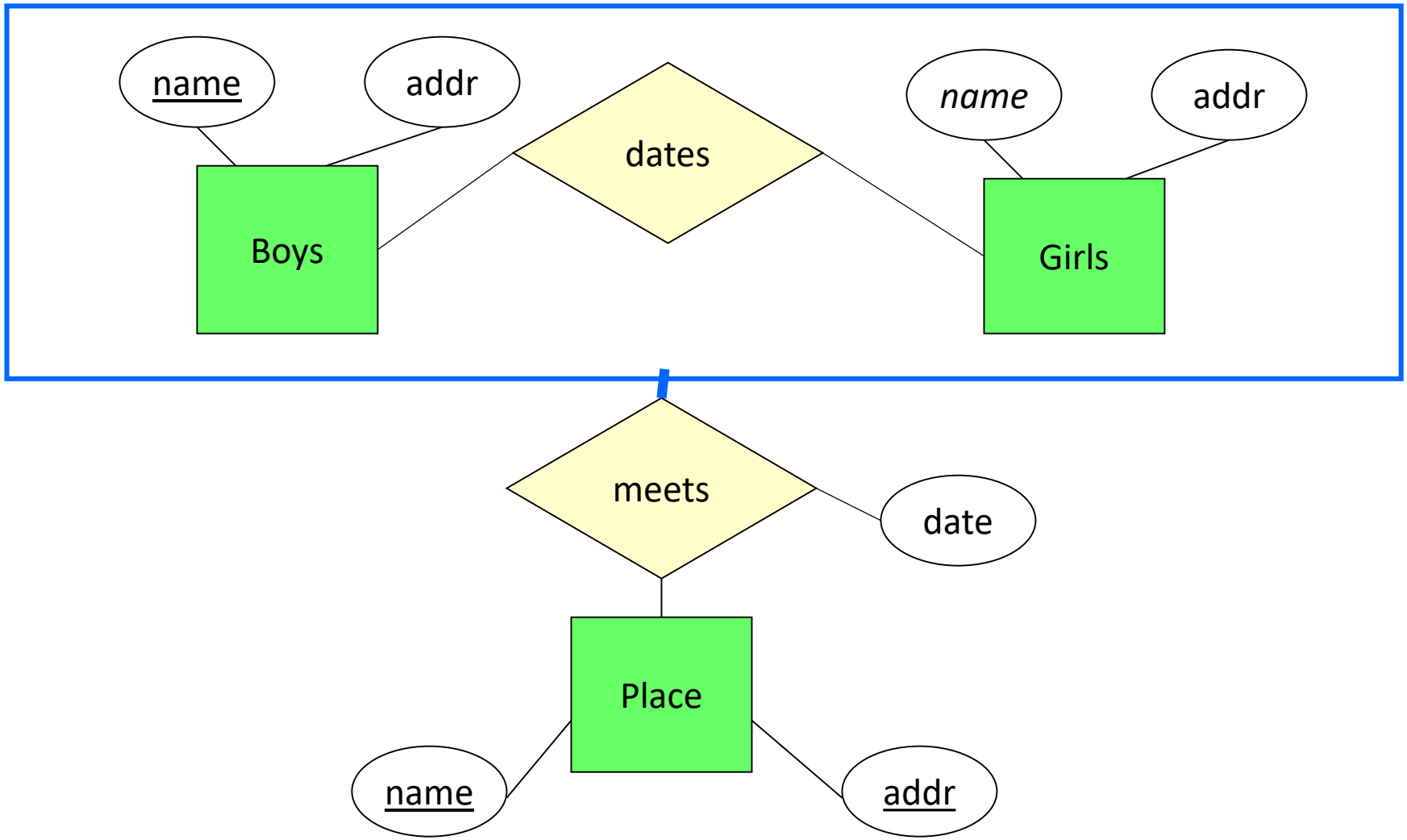


Dates relationship already defines the pair ids in *meets*

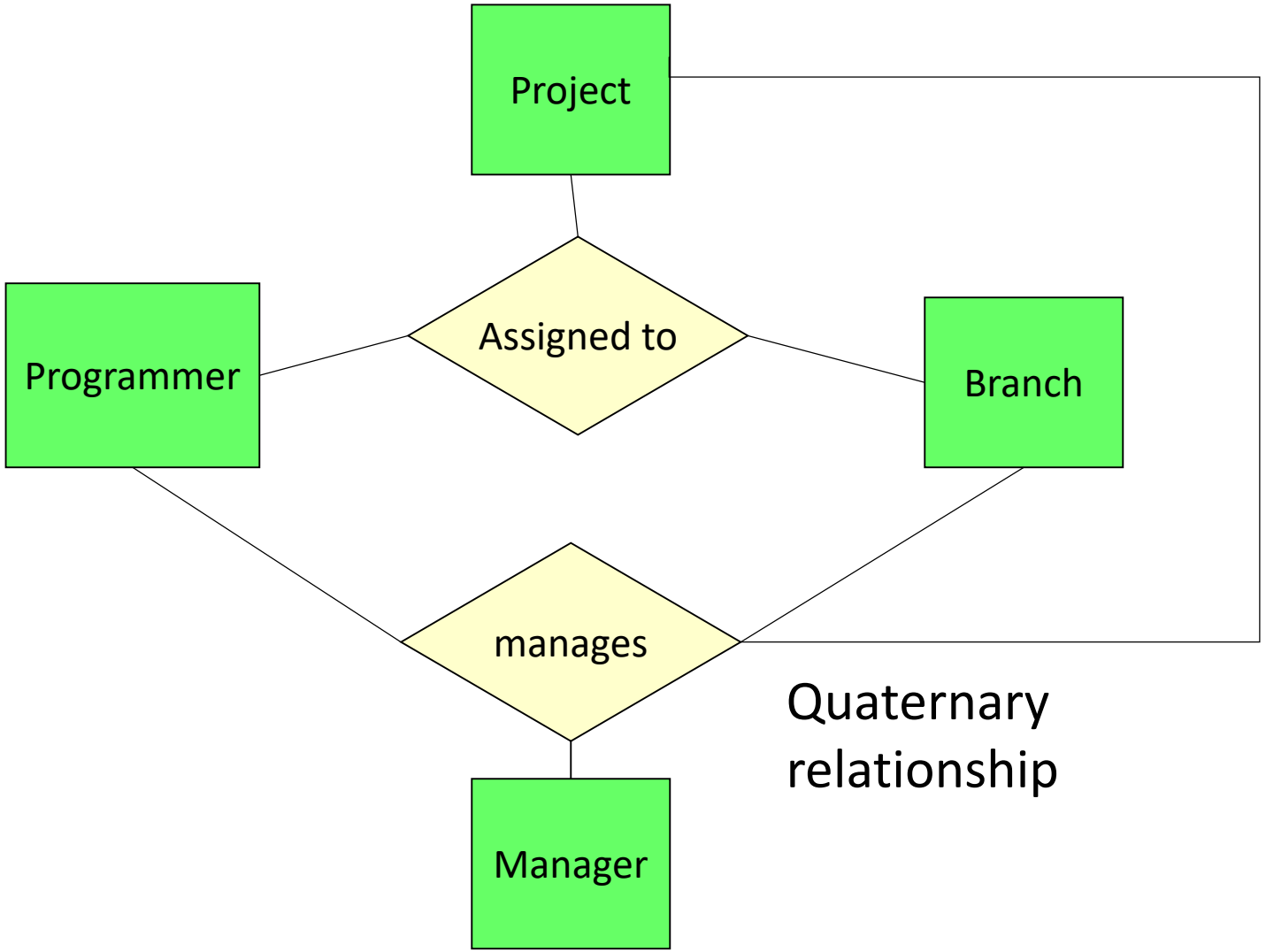


We want to connect 2 relationships – but this is not allowed – they do not unique entities!

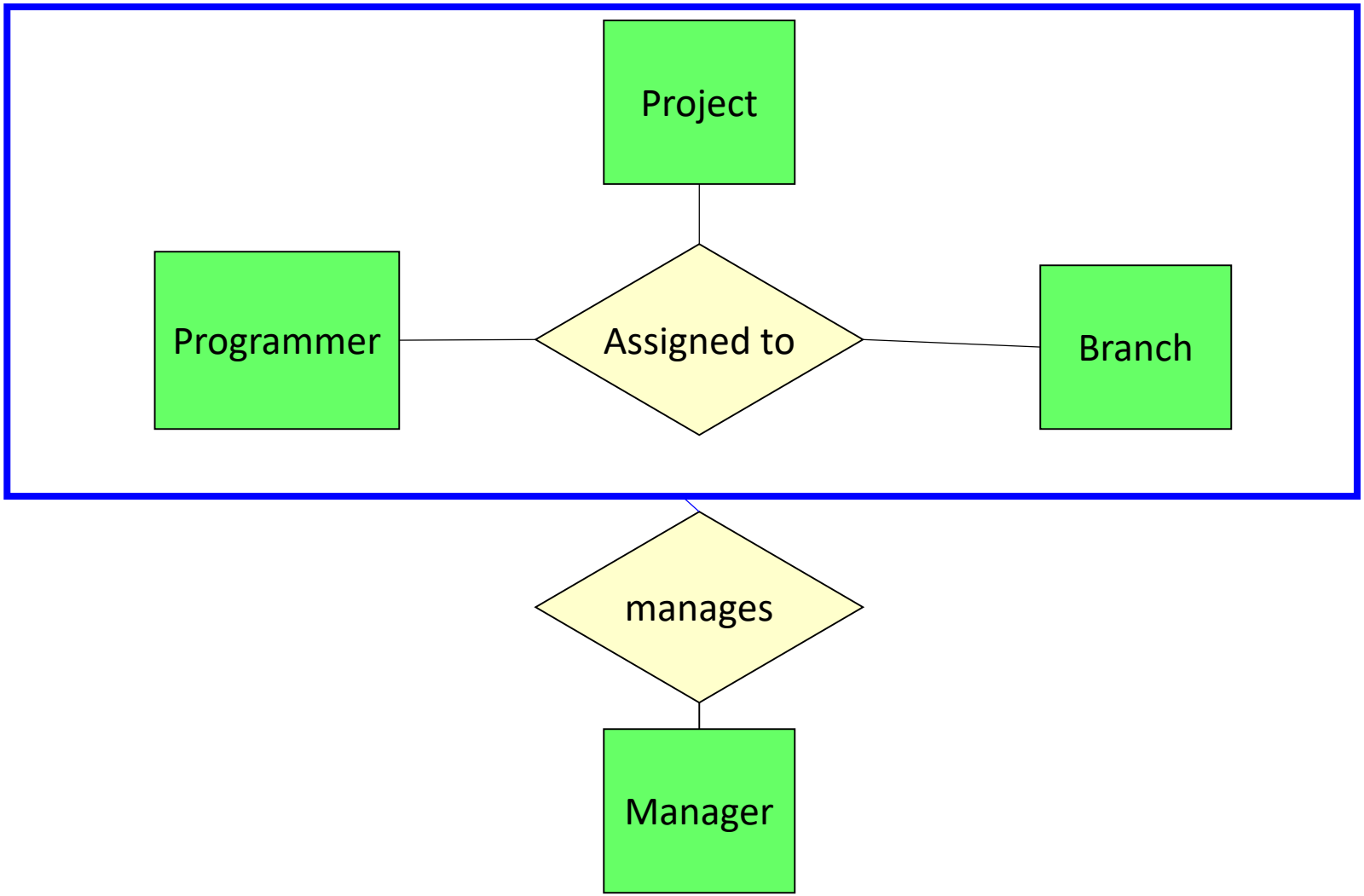
We abstract the info into an aggregate entity



Aggregation: example 2



Abstract each assignment into aggregated entity



E/R notation: last notes

- Limitations of the ER Model:
  - A lot of data semantics can be captured but some cannot (such as functional dependencies)
- Key to successful model: parsimony
  - As complex as necessary, but no more
  - Choose to represent only "relevant" things